Cycle de release LoGeAsWeb

Document de référence — pièce du dossier de certification qualité NF Logiciel. Périmètre : dépôt `logeas-web` (front-end Angular). Dernière mise à jour : 2026-08-03.

Ce document décrit le dispositif maîtrisé de publication de LoGeAsWeb : de la sauvegarde du travail individuel jusqu'à la livraison d'une version au client. Il a vocation à servir de pièce justificative démontrant un processus de release reproductible, tracé et outillé de garde-fous automatiques, conformément aux exigences de maîtrise du cycle de vie logiciel attendues dans le cadre de la certification NF Logiciel.

---

1. Contexte et objectifs

1.1 Constat avant formalisation

Avant la mise en place de ce dispositif, le cycle de publication reposait entièrement sur des gestes manuels, variables selon la personne et le moment :

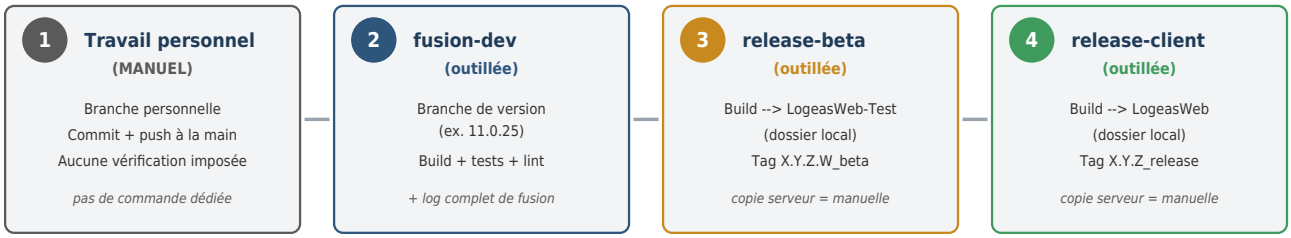
- Aucun garde-fou automatique : rien n'empêchait de poser un tag de release sur un build cassé ou avec des tests en échec.
- Dérive silencieuse du numéro de version : `package.json` était resté bloqué à `11.0.23` alors que la branche de développement en cours s'appelait déjà `11.0.25`, sans que cet écart soit détecté.
- Aucune isolation des builds de test : la configuration de build ne permettait de générer que vers le dossier de livraison client (`LogeasWeb`), y compris pendant une phase de test — aucun dossier `LogeasWeb-Test` distinct n'était réellement alimenté.
- Convention de tag non fiabilisée : coquille orthographique reproduite sur plusieurs tags historiques (`realease` au lieu de `release`).
- Absence de traçabilité fine des fusions : les commits de fusion ne portaient qu'un titre générique, sans lister les commits réellement intégrés.

1.2 Objectifs du dispositif

- **Traçabilité** : chaque étape clé (intégration, beta, release) laisse une trace exploitable (commit détaillé, tag Git, fichier `version.ts` embarquant branche/commit/date de build).
- **Reproductibilité** : les mêmes étapes techniques sont exécutées de la même façon à chaque cycle, indépendamment de la personne qui les déclenche.
- **Garde-fous automatiques** : impossible de publier une beta ou une release sur un code qui ne compile pas ou dont les tests échouent.
- **Séparation stricte des environnements de sortie** : un build de test n'atterrit jamais dans le dossier destiné au client, et réciproquement.
- **Discipline de numérotation** : un numéro de version publié doit pouvoir être retrouvé sans ambiguïté dans l'historique Git, sans numéro « orphelin » ni collision.

2. Vue d'ensemble du cycle

Le cycle se déroule en 4 étapes séquentielles. Seules les étapes 2 à 4 correspondent à une procédure outillée (une commande dédiée, exécutée via l'assistant Claude Code) ; l'étape 1 reste volontairement manuelle. Aucune des procédures outillées ne crée de nouvelle branche Git pour la beta ou la release : elles posent un simple repère (tag) sur la branche de version, conformément à la pratique déjà en usage dans l'équipe.



Aucune étape ne crée de nouvelle branche Git pour la beta ou la release — un simple tag marque le code correspondant.

NB : SVG stocké sur le wiki éditable par exemple avec Inkscape

Étape	Type	Déclencheur	Résultat
1. Travail personnel	Manuel	En continu pendant le développement	Sauvegarde de l'avancement sur une branche personnelle
2. fusion-dev	Outillée	Travail perso jugé prêt à intégrer	Intégration à la branche de version commune, 1ère vérification technique du cycle, log complet des commits fusionnés
3. release-beta	Outillée	Branche de version jugée stable pour test interne	Build déposé dans `LogeasWeb-Test`, tag `X.Y.Z.W_beta`
4. release-client	Outillée	Beta validée (ou release directe si pas de beta jugée nécessaire)	Build déposé dans `LogeasWeb`, tag `X.Y.Z_release`

Le passage d'une étape à l'autre reste une décision humaine (quand fusionner, quand passer en beta, quand livrer). Les procédures exécutent les vérifications techniques une fois la décision prise ; elles ne la prennent pas à la place de l'équipe.

3. Détail des procédures

3.1 Étape 1 — Travail personnel (manuel)

Déclencheur : en continu, à chaque fois qu'on veut sauvegarder son avancement personnel, sans que le travail soit terminé.

Description : chacun committe et pousse sa branche personnelle à sa propre façon. Aucune

commande dédiée, aucune vérification imposée à ce stade (le code peut très bien ne pas compiler ou casser des tests — ce n'est pas grave puisque c'est encore un travail individuel non intégré).

Justification du choix de ne rien outiller ici : imposer un build/tests/lint à chaque sauvegarde intermédiaire ralentirait inutilement un geste censé être rapide et fréquent. Le premier point de contrôle réel du dispositif est l'intégration (étape 2).

Non couvert par cette étape : aucune fusion vers le travail commun de l'équipe — c'est l'objet de l'étape suivante.

—

3.2 Étape 2 — fusion-dev (procédure outillée)

Déclencheur : le travail sur la branche personnelle est jugé prêt à être intégré au tronc commun de la version en cours de développement.

Étapes détaillées :

1. **Identification de la branche cible** : la branche de version numérotée actuellement en développement (à ce jour `11.0.25`), et non `develop` — les cycles récents fusionnent en cascade d'une branche numérotée à la suivante (`11.0.24` → `11.0.25`). En cas de doute sur la branche d'intégration courante, la question est posée explicitement plutôt que devinée.
2. **Mise à jour** : `git fetch`, `git checkout <branche-de-version>`, `git pull` pour être à jour avec `origin` avant de fusionner.
3. **Fusion avec log complet** : récupération de la liste complète des commits qui vont être intégrés (`git log <branche-de-version>..<branche-perso> –pretty=format:"- %s (%h)"`), puis `git merge –no-ff <branche-perso>` avec un message de commit étoffé : le titre standard (`Merge branch 'X' into Y`) suivi en corps de commit de la liste des commits récupérée. Le commit de fusion porte ainsi à lui seul la trace complète de ce qui vient d'être intégré. En cas de conflit à portée métier (logique applicative, pas juste du formatage), validation humaine systématique avant de trancher.
4. **Vérification post-fusion** : `ng build`, `npm test` et `npm run lint` relancés **sur la branche de version après fusion** — l'intégration de plusieurs travaux peut faire apparaître des problèmes invisibles avant fusion. C'est la première vérification technique réelle du cycle.
5. **Publication** : si tout passe, `git push` de la branche de version, log complet inclus. Si un problème est détecté, la publication est bloquée jusqu'à correction (ou `git revert` du merge si le problème est trop complexe à traiter dans la foulée).

Garde-fous : jamais de `push –force` sur cette branche (partagée par toute l'équipe) ; jamais de publication si le build échoue ou si des tests échouent.

Sortie : branche de version à jour, intégrée, vérifiée, avec un historique de fusion exhaustif.

—

3.3 Étape 3 — release-beta (procédure outillée)

Déclencheur : la branche commune de la version en cours est jugée stable et prête pour une phase de test/validation interne.

Étapes détaillées :

1. **État des lieux** : vérification que la branche de version est propre (rien en attente non commité) et à jour avec `origin`. Si des changements traînent, ils sont d'abord traités via l'étape 2 — cette procédure ne fait pas de fusion.
2. **Build** : `npm run build:beta`, qui enchaîne `prebuild:beta` (`node scripts/version.js beta` — voir section 4 pour la règle de numérotation, régénération de `src/assets/version.ts` avec branche/commit/date, régénération de la licence DevExtreme) puis `ng build –configuration production,beta`. La configuration Angular `beta` route la sortie vers `D:/Developpements/VersionPublic/LogeasWeb-Test/` au lieu de `LogeasWeb/` — seule différence technique avec un build release, le reste (optimisations, `environment.prod.ts`) est identique à la configuration `production`. Un build cassé bloque systématiquement la suite.
3. **Commit du bump de version** : le bump (`package.json` + `src/assets/version.ts`) est commité séparément du contenu fonctionnel, message du type `Preparation version X.Y.Z.W`.
4. **Tag** : tag annoté `X.Y.Z.W_beta` sur ce commit, poussé vers `origin` avec le commit.

Garde-fous : jamais de tag sur un build cassé ; jamais de `push –force`.

Sortie : build compilé/optimisé déposé dans `D:\Developpements\VersionPublic\LogeasWeb-Test\browser\`, tag Git correspondant pour toute l'équipe.

Hors périmètre : le déploiement vers le serveur de test réel reste **manuel** — `LogeasWeb-Test` n'est qu'un dossier intermédiaire local, à copier à la main vers le serveur (accessible uniquement en bureau à distance, sans accès réseau direct depuis le poste de développement).

—

3.4 Étape 4 — release-client (procédure outillée)

Déclencheur : la beta correspondante a été validée (ou, si aucune phase beta n'est jugée nécessaire, directement depuis la branche de version), pour produire la version destinée à être livrée au client.

Étapes détaillées :

1. **État des lieux** : vérification que la branche de version est propre et à jour. Si cette release fait suite à une beta, vérification que rien d'autre que des correctifs ciblés n'a été ajouté depuis le tag `_beta` correspondant (`git log <tag_beta>..HEAD`) — sinon un nouveau cycle beta est recommandé avant la release client.
2. **Build** : `npm run build`, qui enchaîne `prebuild` (`node scripts/version.js release`) puis `ng build –configuration production`. La configuration `production` route déjà la sortie vers `D:/Developpements/VersionPublic/LogeasWeb/` (comportement par défaut de `outputPath` dans `angular.json`), pas de configuration supplémentaire nécessaire ici. Un build cassé bloque systématiquement la suite.
3. **Commit du bump de version** : commit séparé du bump (`package.json` + `src/assets/version.ts`), message du type `Preparation version X.Y.Z`.
4. **Tag** : tag annoté `X.Y.Z_release` sur ce commit (orthographe correcte, contrairement aux

anciens tags `_realease`), poussé vers `origin` avec le commit.

Garde-fous : jamais de tag sur un build cassé ; jamais de `push –force`.

Sortie : build compilé/optimisé déposé dans `D:\Developpements\VersionPublic\LogeasWeb\browser\`, tag Git correspondant.

Hors périmètre : le déploiement vers le serveur client reste **manuel**, pour la même raison d'accès que la beta (bureau à distance uniquement).

---

4. Numérotation des versions

4.1 Règle

Le numéro de version comporte 3 ou 4 chiffres selon le type de génération :

- Une **beta** ajoute un **4e chiffre** (`X.Y.Z.W`) qui numérote les tentatives de beta successives au sein d'un même cycle, sans jamais toucher au `X.Y.Z` tant qu'aucune release n'a eu lieu.
- Si le nom de la branche de version ressemble à un numéro (ex. `11.0.25`) et qu'il est supérieur au `X.Y.Z` actuellement dans `package.json`, la première beta du cycle saute directement dessus (rattrapage, sans incrément) et démarre à `W=1`. Ce mécanisme corrige toute dérive entre le nom de la branche et le numéro réellement utilisé par les builds.
- Une **release** repasse en 3 chiffres en retirant simplement le `W`. Elle **reprend le même `X.Y.Z`** que les betas de son cycle : un numéro `X.Y.Z` ne doit jamais rester « orphelin », c'est-à-dire utilisé par des betas sans jamais être publié comme release.
- Le 3e chiffre n'est incrémenté par une release que si ce `X.Y.Z` a déjà été publié par une release antérieure (vérification automatique par consultation des tags Git existants, cas d'une relivraison sans nouvelle beta — par exemple un correctif urgent republié depuis la même branche).

Cette règle garantit qu'il n'existe jamais de numéro de version « perdu » : tout numéro utilisé par une ou plusieurs betas devient, sauf collision, le numéro de la release qui les conclut.

4.2 Implémentation technique

La règle est implémentée dans `scripts/version.js` (fonction `resolveNextVersion`), invoquée avec le mode `beta` ou `release` selon le script npm utilisé :

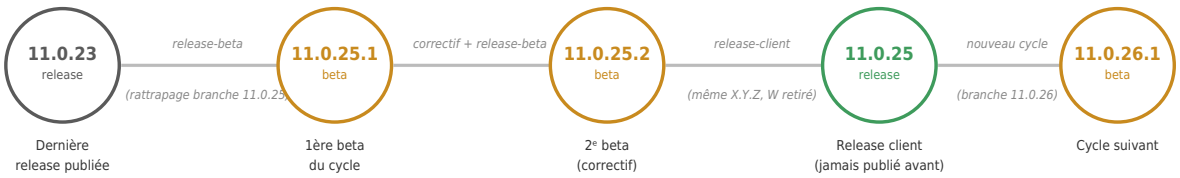
```
function resolveNextVersion(currentVersion, branchName, mode) {
  const [x, y, z, w] = parseVersion(currentVersion);
  const current = [x, y, z];
  let target = current;

  const branchMatch = /^(\d+)\.(\d+)\.(\d+)$/.exec(branchName);
  if (branchMatch) {
    const branchTriplet = branchMatch.slice(1).map(Number);
    if (compareTriplets(branchTriplet, current) > 0) {
```

```
      target = branchTriplet; // rattrapage : nouveau cycle, jamais utilisé
    }
  }
  const isNewCycle = compareTriplets(target, current) > 0;

  if (mode === 'beta') {
    return isNewCycle ? `${target.join('.')}.1` : `${current.join('.')}.${w
+ 1}`;
  }

  // mode === 'release'
  if (isNewCycle) {
    return target.join('.'); // jamais publié : on le prend tel quel
  }
  if (releaseTagExists(current)) {
    return `${x}.${y}.${z + 1}`; // déjà publié : on avance
  }
  return current.join('.'); // reprend le X.Y.Z déjà utilisé par les betas
du cycle
}
```

Script npm	Commande exécutée	Mode
`npm run build:beta`	`node scripts/version.js beta` puis `ng build –configuration production,beta`	beta
`npm run build`	`node scripts/version.js release` puis `ng build –configuration production`	release

4.3 Exemple déroulé sur un cycle complet

Point de départ : la dernière release livrée est `11.0.23` (tag `11.0.23_release` déjà existant). La branche de version du cycle suivant, `11.0.25`, existe déjà et contient du travail non encore livré — `package.json` est donc en retard de 2 par rapport au nom de la branche.



NB : SVG stocké sur le wiki éditable par exemple avec Inkscape

(fichier source : `numerotation-frise.svg`, à téléverser dans le gestionnaire de médias du wiki avant que l'image ne s'affiche)

Étape	Procédure	Branche	package.json (avant → après)	Tag posé

Étape	Procédure	Branche	package.json (avant → après)	Tag posé
1	Travail perso (manuel, plusieurs commits)	feature/import-csv	11.0.23 → 11.0.23 (inchangé)	—
2	fusion-dev (1ère vérification build/tests/lint, log complet)	11.0.25	11.0.23 → 11.0.23 (inchangé)	—
3	release-beta (1ère beta)	11.0.25	11.0.23 → 11.0.25.1 (rattrapage sur la branche, W démarre à 1)	11.0.25.1_beta
4	Travail perso + fusion-dev (correctif suite retour de test)	11.0.25	11.0.25.1 → 11.0.25.1 (inchangé)	—
5	release-beta (2e beta)	11.0.25	11.0.25.1 → 11.0.25.2 (W incrémenté, X.Y.Z inchangé)	11.0.25.2_beta
6	release-client	11.0.25	11.0.25.2 → 11.0.25 (W retiré, même X.Y.Z que les betas — jamais publié avant)	11.0.25_release
7	release-beta (cycle suivant, branche 11.0.26)	11.0.26	11.0.25 → 11.0.26.1 (rattrapage sur la nouvelle branche)	11.0.26.1_beta

Lecture : l'étape 1 (travail perso) ne fait l'objet d'aucune vérification technique — c'est l'étape 2 (fusion-dev) qui vérifie build/tests/lint pour la première fois. Le numéro de version ne bouge qu'aux étapes de génération (release-beta, release-client). Les betas successives d'un même cycle (étapes 3 et 5) ne font varier que le 4e chiffre. La release finale (étape 6) reprend exactement le X.Y.Z des betas qui l'ont précédée puisqu'il n'avait jamais été publié — aucun numéro n'est « sauté » ou laissé inutilisé. Le cycle suivant (étape 7) reprend logiquement le tout prochain numéro (11.0.26, pas 11.0.27 ou plus) : une fois la dérive historique corrigée (voir 1.1), la convention veut que le nom d'une nouvelle branche de version reprenne toujours le numéro immédiatement suivant la dernière release publiée, pour ne plus jamais laisser de trou dans la séquence des numéros livrés aux utilisateurs. Si une release devait à l'inverse être regénérée sur `11.0.25` sans nouvelle beta (le tag `11.0.25_release` existant déjà), ce même numéro `11.0.26` serait alors utilisé pour cette relivraison plutôt que pour un nouveau cycle — les deux cas sont exclusifs l'un de l'autre.

---

5. Garde-fous et traçabilité (aspects qualité)

Ces garde-fous s'appliquent uniformément aux 3 procédures outillées (fusion-dev, release-beta, release-client) :

- **Aucune action destructive automatique** : jamais de `push --force`, jamais de suppression de branche ou de tag. En cas de problème (conflit ambigu, échec de test, divergence avec le serveur distant), la procédure s'arrête et signale l'anomalie plutôt que de forcer une résolution automatique.
- **Blocage systématique sur échec** : un build cassé ou des tests en échec bloquent la suite de la procédure — impossible de produire une beta ou une release sur du code non fonctionnel. Un échec préexistant et sans rapport avec le changement en cours est explicitement signalé plutôt que masqué ou corrigé silencieusement.

- **Traçabilité du code livré** : `src/assets/version.ts`, régénéré à chaque build, embarque la branche Git, le hash de commit court et la date de build — chaque livraison (beta ou release) est ainsi reliée sans ambiguïté à un état précis du code source.
- **Traçabilité des fusions** : depuis la mise à jour de fusion-dev, chaque commit de fusion porte en corps de message la liste complète des commits individuels intégrés, et non plus seulement un titre générique.
- **Traçabilité des publications** : chaque beta et chaque release est marquée par un tag Git annoté et poussé, permettant de retrouver exactement quel commit correspond à quelle publication.
- **Discipline de numérotation** : voir section 4 — aucun numéro de version publié ne peut être réutilisé (vérification par consultation des tags existants), et aucun numéro n'est utilisé par une beta sans finir par être publié en release.

---

6. Ce qui reste volontairement manuel

- **Le déploiement final vers les serveurs** (test et client) : `LogeasWeb` et `LogeasWeb-Test` sont des dossiers intermédiaires locaux, copiés à la main vers le serveur réel. Ces serveurs ne sont accessibles que par connexion à distance (bureau à distance), sans accès réseau direct depuis le poste de développement — la copie ne peut donc pas être automatisée dans le cadre de ce dispositif.
- **La sauvegarde du travail personnel** (étape 1) : choix délibéré pour ne pas ralentir les sauvegardes fréquentes d'un travail potentiellement inachevé (voir 3.1).
- **La décision de progression** : quand fusionner, quand passer en beta, quand livrer au client — ce sont des décisions d'équipe, jamais automatiques. Les procédures exécutent les vérifications techniques une fois la décision prise.

---

7. Évolutions apportées par ce dispositif

Avant	**Après**
Étapes manuelles, non reproductibles	3 procédures outillées (fusion-dev, release-beta, release-client), identiques à chaque exécution
Aucun garde-fou : possibilité de tagger un build cassé	Blocage systématique si le build échoue ou si des tests échouent
`package.json` dérivé du nom des branches (23 vs 25) sans détection	Rattrapage automatique sur le nom de la branche de version
Build de test et build client indifférenciés (tout partait vers `LogeasWeb`)	Configuration Angular `beta` dédiée, sortie routée vers `LogeasWeb-Test`
Numéro de beta et de release parfois désynchronisés du nom de branche, betas consommant un numéro jamais publié	Beta = 4e chiffre, release = reprise du même 3e chiffre — aucun numéro orphelin
Coquille orthographique reproduite sur les tags (`realease`)	Orthographe correcte (`release`) sur les nouveaux tags ; anciens tags non renommés
Commit de fusion au titre générique, sans détail	Commit de fusion enrichi du log complet des commits intégrés

8. Annexes techniques

8.1 Fichiers concernés (dépôt logeas-web)

Fichier	Rôle
`scripts/version.js`	Calcul et application de la règle de numérotation (section 4)
`angular.json` (configuration `beta` du builder `build`)	Redirection de la sortie de build vers `LogeasWeb-Test` en mode beta
`package.json` (scripts `build`, `build:beta`, `prebuild`, `prebuild:beta`)	Points d'entrée des générations beta et release

8.2 Procédures outillées (assistant Claude Code)

Commande	Fichier source	Rôle
`/fusion-dev`	`.claude/commands/fusion-dev.md`	Étape 2 : intégration + vérification + log complet
`/release-beta`	`.claude/commands/release-beta.md`	Étape 3 : build beta + tag
`/release-client`	`.claude/commands/release-client.md`	Étape 4 : build release + tag

8.3 Convention de nommage des tags Git

- Beta : `X.Y.Z.W_beta` (ex. `11.0.25.1_beta`)
- Release : `X.Y.Z_release` (ex. `11.0.25_release`)

---

9. Points ouverts à valider avec l'équipe

- Le passage en beta n'est pour l'instant jamais sauté avant une release client de façon automatique — à confirmer que c'est bien la règle voulue dans tous les cas.
- La convention de nommage des branches de version (ex. `11.0.25`) suppose de connaître le numéro cible à l'avance — à confirmer que c'est toujours décidé en amont et non ajusté en cours de cycle.
- Les noms des procédures pourront être ajustés une fois éprouvés en pratique, avant validation définitive de cette page pour le dossier de certification.

Document généré à partir de l'état du dépôt `logeas-web` au 2026-08-03. À revalider après toute évolution du dispositif décrit ci-dessus.