Release Strategy

Unified versioning and release management

Current: Main Branch

- Developer merges PR to main with conventional commits
- release-please analyzes commits since last release
- Determines version bump: `feat:` → minor, `fix:` → patch, `BREAKING CHANGE:` → major
- Opens/updates release PR (updates CHANGELOG.md and package.json)
- Merge creates prerelease tagged `v1.25.0-beta`
- Triggers `release/X.Y.x` branch creation
- Empty commit with `Release-As: X.Y.0` removes `-beta` suffix

Current: Hotfix Workflow

- Push to `release/**` branch
- release-please auto-bumps patch
- Creates release PR
- Merge creates stable release

Current Challenges

Independent versioning per workspace

Package.json: 1.25.0-beta

Controls apps: 0.1.0

Firmware: 0.1.0

Explorer: 1.0.0

No single version number represents the suite of software

Benefits Worth Preserving

Automated release process

Conventional commit based

Changelog generation included

Separate workflows for main and release branches

New Strategy: Unified Versioning

All version strings synced:

- package.json at repo root
- controls workspace Cargo.toml
- firmware workspace Cargo.toml
- explorer workspace Cargo.toml
- cabinet-controller Cargo.toml

Release Process

When we want to make a release:

- Bump all version strings uniformly, update changelog, commit on main
- Create release branch - tags commit on main (e.g., `v1.26.0`) and creates branch `release/1.26.x`
- Switch to release branch and publish - creates GitHub release from the tag

Hotfix Process

To apply hotfixes to release branches:

- Cherry-pick the SHA of the commit and bump the patch version
- Publish creates new tag on release branch (e.g., `v1.26.1`) and GitHub release

What We Get

- Uniform semantic versioning
- Tags on main branch commits
- Release branches made from those tagged commits
- Ability to add hotfixes to release branches and publish new releases including them

The makeline-release Tool

Custom tool called makeline-release handles the release workflow

Just commands wrap each step

Replaces release-please

Commands: Version Bumps

```
# Update version in all 5 places
# And update CHANGELOG.md at repo root
just bump-minor-version          # 1.25.0 → 1.26.0
just bump-major-version          # 1.25.0 → 2.0.0
```

CHANGELOG.md automatically updated during bump step

Run on main only - release branches only get patch bumps via hotfix

Commands: Create Release Branch

```
# Tag current commit and create release branch
just create-release-branch       # Tag: v1.26.0, Branch: release/1.26.x

# Or with suffix (for variants/pre-releases)
just create-release-branch beta  # Tag: v1.26.0-beta, Branch: release/1.26.x-beta
```

How Tagging Works

On main at commit `abc123` with version `1.26.0`:

- Creates tag `v1.26.0` pointing to `abc123` on main
- Creates branch `release/1.26.x` from `abc123`
- Tag on main and first commit on release branch are the same

Tag marks the point in main's history where the release was cut

Commands: Publish Release

```
# You're already on the release branch after create-release-branch
# Initial release uses existing tag from main
just publish-release                    # GitHub release v1.26.0
```

Development continues on main

Commands: Hotfixes

```
# On release branch: find commits to backport
git log main --oneline

# Apply hotfix - cherry-picks and bumps patch version
just hotfix <sha>

# Publish release with hotfixes - creates new tag on release branch
just publish-release                    # Tag: v1.26.1, GitHub release created
```

Commands: Dry Run Mode

You can append -dry to any command:

```
just bump-minor-version-dry
just bump-major-version-dry
just create-release-branch-dry
just publish-release-dry
```

Release Validation

- Explorer displays unified version
- Validation widget verifies deployed binaries
- `just generate-hashes` creates executables.json

```
{
  "version": "1.25.0",
  "executables": {
    "api": "0b4e6bf10c7a63ac...",
    "lifecycler_server": "b52551483b0dd6e1...",
    ...
  }
}
```

CI Integration

Publishing with `release/**` branch name still triggers existing workflows:

- Firmware artifact build and upload
- Greengrass component compilation
- Greengrass component deployment
- Artifact uploading to GitHub release

Benefits

- Clear compatibility - single version tells you what works together
- Automated changelogs and version updates
- Development decoupled from releases (no release PRs)
- Explicit control over when to release
- Support for release variants (beta, hardware-specific, etc.)
- Release validation via hash verification
- Tags on main mark branch points, releases published from release branches

Questions?