0-deb: A Container-First Linux Distribution

Technical Whitepaper | Version 1.0

Executive Summary

0-deb is a container-first Linux distribution inspired by Debian, engineered specifically for ephemeral container environments. Unlike general-purpose distributions that retrofit container support onto desktop-oriented foundations, 0-deb is built from first principles around container semantics: images are immutable, installations are always fresh, and runtime footprint must be minimal.

The distribution is powered by two purpose-built tools:

Debflow — A software supply chain system that builds Debian-compatible packages from upstream source with bit-for-bit reproducibility, multi-source vulnerability tracking, and container-optimized maintainer scripts. Debflow produces packages with enhanced metadata for transparent CVE identification and publishes a security feed for accurate vulnerability scanning.

Captain (capt) — A drop-in replacement for apt designed exclusively for container build phases. Captain uses the Pubgrub algorithm for dependency resolution, supports object storage backends for package archives, and leaves zero footprint in final images through its `capt goodbye` command.

Together, these components deliver a distribution where every package is reproducible, every vulnerability is accurately tracked, and every container image contains only what's necessary for the workload.

1. The Container Packaging Problem

Traditional Linux distributions carry assumptions incompatible with container deployment models. This section examines the fundamental mismatches that 0-deb addresses.

1.1 Desktop-Oriented Package Defaults

Debian maintainer scripts assume persistent systems with init managers, interactive operators, and upgrade paths. They invoke *systemctl* for service management, *debconf* for interactive configuration, and version comparison logic for migrations. Containers are ephemeral—they're replaced entirely, not upgraded in place. These assumptions cause unnecessary complexity.

1.2 Package Manager Overhead

Traditional package managers like apt are designed for long-lived systems. They maintain state databases, cache metadata, and support upgrade/removal operations irrelevant to container builds. This infrastructure persists in final images, consuming space and expanding the attack surface.

1.3 Vulnerability Attribution Across Naming Conventions

The National Vulnerability Database (NVD) and Debian use different naming conventions for identical software. Security scanners querying NVD encounter systematic mismatches, e.g: Oracle's Berkleydb is packaged as db5.3. Also, scanners can only rely on Debian's security feed as vendor data is also not provided in package metadata.

---

2. 0-deb:

2.1 Components

0-deb Linux Distribution	
Debflow (Package Build System)	**Captain** (Package Manager)
● Reproducible Builds	● Pubgrub package solver
● Rolling Release	● S3-compatible backends
● Branch-aware CVE patching	● Zero footprint cleanup
● Agentic script transformation	● Install-only semantics
● Minimal Dependency chains	● Drop-in apt replacement
● Multi arch: amd64 + arm64	

2.2 Design Principles

Container-First: Every design decision optimizes for ephemeral, immutable container images and deployment practices. Features irrelevant to this model are intentionally omitted.

Minimal Dependencies: Packages are built with reduced dependency chains, eliminating optional components unnecessary for container workloads.

Zero Footprint: The package manager removes all traces of itself from final images, leaving only installed software.

Single Rolling Release: 0-deb maintains one rolling release channel suited to container builds. No need to track multiple distribution versions or coordinate upgrades across releases.

Transparent Security: Enhanced metadata enables vulnerability scanners to accurately identify CVEs without manual mapping configuration.

---

3. Captain: The Container Package Manager

Captain (capt) is a purpose-built package manager for container builds, designed as a drop-in replacement for apt.

3.1 Design Philosophy

Install-only: Package installation is the only supported operation. Deletion and upgrade are intentionally not implemented—containers are replaced, not modified.

Zero Footprint: After installation, capt goodbye removes all Captain artifacts including itself, leaving only installed packages.

Build-Phase Tool: Captain exists only during image build. It never runs in deployed containers.

3.2 Zero-Footprint Cleanup

The capt goodbye command removes all traces of Captain:

What capt goodbye Removes	What Remains
Captain binary	Installed package files
Package metadata and indices	Created directories and symlinks
Download cache	File permissions set by packages

Configuration files	Installed packages metadata

3.3 Drop-in apt Replacement

Apt Command	Capt Equivalent	Notes
apt update	capt update	Fetch package indices
apt install pkg	capt install pkg	Install packages
apt install pkg=ver	capt install pkg=ver	Install specific package version
apt remove pkg	-	Not implemented
apt upgrade	-	Not implemented

Dockerfile migration is typically mechanical:

None
```
# Before (apt)
RUN apt-get update && apt-get install -y nginx python3 \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# After (capt)
RUN capt update && capt install nginx python3.13 && capt
goodbye
```

3.4 Object Storage Backend Support

Captain supports S3-compatible object storage for package archives:

Shell
```
# Koala.sources file to be consumed by capt
deb [arch=amd64] s3://0deb-packages/0deb testing main
```

3.5 Pubgrub Dependency Resolution

Captain uses the Pubgrub algorithm for version solving, providing:

Optimal Solutions: Finds solutions satisfying all constraints or proves none exist

Clear Error Messages: When resolution fails, explains exactly which constraints conflict

Performance: Efficient backtracking for complex dependency graphs

Determinism: Same inputs always produce same resolution

3.6 Why No Remove or Upgrade?

Captain intentionally omits package removal and upgrade:

Remove: In containers, if you don't want a package, don't install it. Multi-stage builds handle cases where build dependencies shouldn't appear in final images. Removal logic adds complexity for a use case better served by build patterns.

Upgrade: Containers are immutable. To "upgrade," rebuild the image with new package versions. In-place upgrades imply a persistent state that containers shouldn't have. The single rolling release means the latest packages are always available for new builds.

---

4. Security Pipeline

Debflow implements a comprehensive security pipeline with automated CVE discovery, branch-aware patching, and security feed publishing.

4.1 Branch-Aware CVE Patching

CVE-2025-12345 is discovered for a package. All versions < 3.5.1 are vulnerable. As a response to this CVE, we will release fixed version 3.5.1 and if necessary and feasible, we will backport the fix to 3.4.x branch in version 3.4.8-1.

Debflow correctly handles vulnerabilities across development branches:

```
fix_versions = ["3.5.1", "3.4.8-1"]
fix_branch("3.5.1") = "3.5"
fix_branch("3.4.1-1") = "3.4"
```

Version	State
3.4.0	Vulnerable
3.4.1-2	Fixed (in 3.4.1-1 for 3.4.x)
3.5.0	Vulnerable as fixed version for 3.5.x branch: 3.5.1 > 3.5.0
3.5.4	Fixed (Versions >= 3.5.1 aren’t vulnerable)
3.6.2	Fixed (Versions >= 3.5.1 aren’t vulnerable)

Status Determination:

Status	Meaning
fixed	Our version >= fix version from same branch
open	No fix available or fix is for different branch
not_affected	Explicitly marked as not vulnerable
needs_review	CVE not in Debian Security Tracker data

4.4 Security Feed

The security feed provides machine-readable vulnerability status:

```json
{
  "package": {
    "CVE-2025-12345": {
      "fixed_in": [{
           "version": "3.5.1",
           "vulnerable_range": ">=3.5.0 <3.5.1"
      },
      {
           "version": "3.4.8-1",
           "vulnerable_range": "<3.4.8-1"
      }]
    },
    "CVE-2025-9232": {
      "fixed_in": []
    }
  },
  "python3.13": {
    "CVE-2024-12718": {
      "fixed_in": [{"version": "3.13.3-1"}]
    },
    "CVE-2024-3220": {
      "fixed_in": [{"version": 0}]
    }
  }
}
```

Feed semantics:

Field	Description
`fixed_in: [{"version": "X"}]`	Fixed in version X
`fixed_in: [{"version": "3.5.4-1", “vulnerable_range”: “>=3.5 <3.5.4-1”}]`	Fixed in version 3.5.4-1 for 3.5.x branch
`fixed_in: []`	Open, no fix available
`fixed_in: [{"version": 0}]`	Not affected

Published at: `https://security.0-deb.dev/feed.json`

4.5 Grype Integration

0-deb publishes a security feed that is consumed by Anchore/Vunnel. When fully integrated with grype, it enables Grype to recognise custom fixes specific to 0-deb.

The injected ZERODEB metadata is used on a custom patched version of Grype that enables scanning of 0-deb’s packages without relying on distro’s security feed:

- Match packages to correct NVD products via embedded vendor/source name fields
- Compare against upstream source versions
- Correlate with 0-deb security feed for accurate status

4.3 Automated Patched Version Packaging

Once a patch is either released upstream or is available in debian sources, the rest of the process is fairly automated to release the fixed version at the earliest.

1. Detects new versions via “*uscan*” monitoring of upstream and Debian
2. Generates updated Debflow manifest(debify.yaml) with new patches, checksums and snapshot sources
3. Applies agentic script transformation to maintainer scripts if it has changed in the debian sources
4. Builds and tests packages for arm64 and amd64
5. Updates security feed with new fix versions
6. Publishes to repository

This mirrors Debian's security response while maintaining 0-deb's container optimizations. In case CVE is fixed upstream first, the pipeline will release a new version.