Выпущен Alpine Linux 3.24.2

Вышел Alpine Linux 3.24.2. Это сервисное обновление стабильной ветки 3.24, без больших новых возможностей, но с важной практической частью: в релизе закрыты известные уязвимости OpenSSL высокой и средней критичности, а так же вошли другие исправления безопасности и ошибок.

Официальный анонс короткий. Alpine Linux 3.24.2 был опубликован 17 сентября 2026 года вместе с обновлениями других поддерживаемых веток: 3.21.8, 3.22.6 и 3.23.6. Проект пишет, что эти выпуски исправляют все известные OpenSSL-уязвимости высокой и средней степени критичности, применимые к соответствующим веткам, плюс включают другие security и bug fixes. Полный список изменений разработчики предлагают смотреть в git log релиза.

На странице загрузок Alpine 3.24.2 указан как текущая версия. Для ветки 3.24 доступны обычные варианты поставки: Standard, Extended, Netboot, Virtual, образы для Raspberry Pi, Generic U-Boot, Xen, а так же Mini root filesystem, который как раз интересен для контейнеров и минимальных chroot-окружений. Сама ветка 3.24 была создана 9 июня 2026 года и поддерживается до 1 июня 2028 года.

Для обычной установленной системы это повод сделать плановое обновление. Для контейнеров ситуация немного интереснее, потому что Alpine часто используется как базовый слой: маленький rootfs, `musl`, BusyBox и быстрый apk хорошо подходят для компактных образов.

Типичный Dockerfile выглядит примерно так:

dockerfile copy

```
FROM alpine:3.24

RUN apk add --no-cache ca-certificates
```

Сам по себе такой Dockerfile еще не означает, что в registry уже лежит образ с Alpine 3.24.2. Если приложение было собрано раньше, оно продолжает использовать старый базовый слой. Его нужно пересобрать и заново проверить.

Проверяем версию в контейнере

Самый простой способ посмотреть, какая версия Alpine сейчас внутри образа:

bash copy

```
docker run --rm alpine:3.24 cat /etc/alpine-release
```

Для актуального образа должно показать:

text copy

```
3.24.2
```

Так же можно проверить `/etc/os-release`:

bash copy

```
docker run --rm alpine:3.24 cat /etc/os-release
```

Там будет видно имя дистрибутива и номер версии. Для своих образов проверка такая же:

bash copy

```
docker run --rm my-app:latest cat /etc/alpine-release
```

Если там осталась старая версия, значит образ не был пересобран на новом базовом слое.

Пересборка своих образов

Если используется обычный тег `alpine:3.24`, обычно достаточно забрать свежий базовый образ и собрать приложение заново:

bash copy

```
docker pull alpine:3.24
docker build --pull -t my-app:latest .
```

Параметр `--pull` полезен тем, что Docker попробует забрать свежий базовый образ, а не использовать то, что уже лежит локально в кеше.

В CI это особенно важно. Dockerfile может выглядеть правильно, но если сборка работает с кешем и не обновляет base image, исправления безопасности в итоговый artifact не попадут.

Про digest

Для продакшена часто закрепляют образ не только по тегу, но и по digest:

dockerfile copy

```
FROM alpine:3.24@sha256:<digest>
```

Это хорошо для повторяемых сборок: сегодня, завтра и через месяц будет использоваться один и тот же базовый образ. Но есть обратная сторона: после выхода security-релиза digest нужно обновлять осознанно.

Если образ закреплен так жестко, Alpine 3.24.2 сам в него не попадет. Нужно поменять digest, пересобрать приложение и прогнать тесты.

Минимальный rootfs

У Alpine есть Mini root filesystem. Это минимальная файловая система для контейнеров и chroot-окружений.

Это полезно, если образ собирается не классическим Dockerfile через `FROM alpine`, а своим способом. Например, есть отдельный pipeline сборки OCI-образа или нужен минимальный rootfs для внутреннего runtime-образа.

Использовать его можно как готовую основу. Скачать Mini rootfs можно с официального CDN Alpine. Для x86_64 ссылка будет такая:

bash copy

```
curl -LO https://dl-cdn.alpinelinux.org/alpine/v3.24/releases/x86_64/alpine-minirootfs-3.24.2-x86_64.tar.gz
```

Для других архитектур меняется часть пути после `releases`: `aarch64`, `armv7`, `ppc64le`, `riscv64`, `s390x` и так далее. Если нужен просто текущий стабильный релиз без привязки к ветке 3.24, можно смотреть каталог `https://dl-cdn.alpinelinux.org/alpine/latest-stable/releases/`.

После скачивания из tarball можно сделать контейнерный образ:

bash copy

```
docker import alpine-minirootfs-3.24.2-x86_64.tar.gz alpine-minirootfs:3.24.2
docker run --rm alpine-minirootfs:3.24.2 cat /etc/alpine-release
```

Другой вариант — положить tarball рядом с Dockerfile и распаковать его в пустой образ:

dockerfile copy

```
FROM scratch
ADD alpine-minirootfs-3.24.2-x86_64.tar.gz /
```

```
CMD ["/bin/sh"]
```

Такой способ удобен когда нужно полностью контролировать базовый слой и не зависеть от готового образа из registry. Перед использованием в нормальном пайплайне лучше проверять sha256 и GPG-подпись, которые Alpine публикует рядом с архивом.

В таком случае 3.24.2 тоже имеет смысл подтянуть отдельно. Обновление находится не только в готовом docker image, но и в самом базовом наборе пакетов.

Пример с Go-программой

Для проверки можно собрать простую программу на Go и положить ее в несколько разных образов.

Цель проверки — посмотреть, сколько места занимает одно и то же статически собранное приложение в трех вариантах окружения:

- обычный официальный образ `alpine:3.24`;
- Alpine Mini rootfs, добавленный из tarball;
- совсем пустой `scratch`-образ, где лежит только исполняемый файл.

Так сразу видно, где заканчивается размер самого приложения и где начинается размер пользовательского окружения: shell, apk, базовые файлы Alpine и все остальное.

Программа будет совсем простой:

go copy

```
package main

import "fmt"

func main() {
        fmt.Println("hello from alpine")
}
```

Сохраним ее как `main.go` и соберем статический бинарник. Для такого примера не нужен CGO, поэтому отключаем его и дополнительно убираем отладочную информацию:

bash copy

```
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -trimpath -ldflags="-s
-w" -o hello main.go
```

Теперь сделаем обычный образ на Alpine:

dockerfile copy

```
FROM alpine:3.24
COPY hello /usr/local/bin/hello
CMD ["/usr/local/bin/hello"]
```

Собираем:

bash copy

```
docker build -f Dockerfile.alpine -t hello-alpine:3.24 .
```

Второй вариант — образ из Mini rootfs. Рядом с Dockerfile должен лежать архив `alpine-minirootfs-3.24.2-x86_64.tar.gz`:

dockerfile copy

```
FROM scratch
ADD alpine-minirootfs-3.24.2-x86_64.tar.gz /
COPY hello /usr/local/bin/hello
CMD ["/usr/local/bin/hello"]
```

Собираем его отдельно:

bash copy

```
docker build -f Dockerfile.minirootfs -t hello-minirootfs:3.24.2 .
```

Третий вариант — совсем без rootfs:

dockerfile copy

```
FROM scratch
COPY hello /hello
CMD ["/hello"]
```

Собирается так:

bash copy

```
docker build -f Dockerfile.scratch -t hello-scratch:go-static .
```

Проверяем запуск:

bash copy

```
docker run --rm hello-alpine:3.24
docker run --rm hello-minirootfs:3.24.2
docker run --rm hello-scratch:go-static
```

Все три контейнера должны вывести:

text copy

```
hello from alpine
```

Размеры можно сравнить так:

bash copy

```
docker image ls hello-alpine:3.24
docker image ls hello-minirootfs:3.24.2
docker image ls hello-scratch:go-static
```

На моей машине получились такие размеры. Проверка выполнялась на `linux/amd64`, Go 1.27.1 и Docker 29.8.1:

Образ	Размер
`hello-alpine:3.24`	9.93 MB
`hello-minirootfs:3.24.2`	9.93 MB
`hello-scratch:go-static`	1.51 MB

В этом тесте обычный `alpine:3.24` и образ из Mini rootfs получились одинаковыми по размеру. Mini rootfs здесь нужен для другого сценария: когда базовую файловую систему берут напрямую из tarball и включают в свой pipeline сборки.

В этом же тесте `hello-scratch:go-static` получился размером 1.51 MB. Это почти размер самого бинарника `hello`, который после сборки занимал 1.5 MB. Разница уже хорошо видна в таблице, но там не будет shell, apk, CA certificates и привычного окружения Alpine. Для некоторых сервисов это отлично, для других слишком аскетично.

Что проверить после обновления

В большинстве случаев переход с 3.24.1 на 3.24.2 должен пройти спокойно. Но после пересборки контейнера стоит проверить хотя бы базовые вещи:

- приложение стартует из чистого образа;
- TLS-соединения работают;
- сертификаты установлены и находятся там, где приложение их ожидает;
- healthcheck проходит;

- сканер уязвимостей смотрит именно собранный образ, а не только Dockerfile.

Отдельно стоит помнить про `musl`. Alpine использует `musl libc`, а не `glibc`. Обычно это не проблема, но если приложение тянет native-модули для Python, Node.js, Ruby или Go с CGO, проверять нужно именно запуск приложения, а не только успешную установку пакетов.

Итого

Alpine Linux 3.24.2 — небольшой, но нужный релиз. Новых больших возможностей тут нет, зато есть исправления безопасности в базовом слое, который потом уезжает во множество контейнеров.

Если где-то используется Alpine 3.24, стоит пересобрать образы, обновить digest если он закреплен, прогнать smoke-тесты и заново посмотреть результат сканера уязвимостей.

Источники: анонс Alpine Linux 3.24.2, страница загрузок Alpine Linux.