Web Push: Что такое и с чем едят

08.09.2026 14:05 edited 17.09.2026 23:47

Введение

Web Push — это технология, позволяющая веб-приложениям получать сообщения от сервера даже тогда, когда страница не загружена в браузере или находится в фоновом режиме. В отличие от классических HTTP-запросов, которые инициируются клиентом, push-уведомления инициируются сервером и доставляются через инфраструктуру браузера, что делает их идеальным инструментом для оповещений, обновлений и re-engagement'a пользователей.
Push API является стандартизированным W3C-спецификацией и с марта 2023 года **широко доступен** во всех основных браузерах. Однако за кажущейся простотой отправки уведомления скрывается сложная экосистема из протоколов, механизмов шифрования, сервис-воркеров и push-сервисов. Мы разберём Web Push до уровня протоколов, рассмотрим архитектуру, безопасность, новые стандарты и практические нюансы реализации.

Архитектура Web Push: три ключевых компонента

Web Push строится на взаимодействии трёх основных сущностей:

1. **Браузер (User Agent)** — клиентская среда, в которой работает веб-приложение.
2. **Push-сервис (Push Service)** — внешний сервис, предоставляемый поставщиком браузера (например, FCM от Google для Chrome, APNs от Apple для Safari), который принимает push-сообщения от сервера приложений и доставляет их на устройства пользователей.
3. **Сервер приложений (Application Server)** — ваш бэкенд, который инициирует отправку push-сообщений.

Ключевая особенность: сервер приложений никогда не взаимодействует с браузером напрямую. Все сообщения проходят через push-сервис, который выполняет роль посредника, обеспечивая доставку даже тогда, когда браузер закрыт или устройство офлайн.

Service Worker — невидимый обработчик

Для получения push-сообщений веб-приложение должно зарегистрировать **service worker**. Service worker — это JavaScript-скрипт, который работает в фоновом режиме, независимо от открытой страницы. Именно он:

- Подписывается на push-уведомления через `PushManager.subscribe()`
- Принимает входящие push-события через обработчик `onpush`
- Отображает уведомления через `ServiceWorkerRegistration.showNotification()`

```
// Регистрация service worker
navigator.serviceWorker.register('/sw.js');

// В service worker (sw.js)
self.addEventListener('push', (event) => {
  const data = event.data.json();
  event.waitUntil(
    self.registration.showNotification(data.title, {
      body: data.body,
      icon: '/icon.png'
    })
  );
});
```

Каждая подписка уникальна для конкретного service worker, а её **endpoint** (URL, на который сервер отправляет сообщения) является **capability URL** — знание этого URL достаточно для отправки сообщения. Именно поэтому endpoint необходимо хранить в секрете.

Протокол Web Push: как сервер отправляет сообщение

Протокол, определяющий формат HTTP-запроса от сервера приложений к push-сервису, описан в **RFC 8030**. Этот запрос — не просто POST с телом сообщения, он включает сложную систему аутентификации и шифрования.

VAPID — аутентификация сервера приложений

VAPID (Voluntary Application Server Identification) — это механизм, позволяющий push-сервису удостовериться, что сообщение отправлено именно тем сервером приложений, который создал подписку. Это предотвращает ситуацию, когда злоумышленник, получивший endpoint, может отправлять сообщения от имени вашего приложения. Процесс аутентификации выглядит так:

1. Сервер приложений генерирует пару ключей VAPID (публичный и приватный) на основе кривой P-256.
2. Публичный ключ передаётся в браузер при вызове `PushManager.subscribe()`.
3. При отправке push-сообщения сервер создаёт **JSON Web Token (JWT)** с полями:

`aud` (audience) — URL push-сервиса
`exp` (expiration) — timestamp истечения срока действия токена
`sub` (subject) — контактный email разработчика

1. JWT подписывается приватным ключом VAPID с использованием алгоритма ES256.
2. Подписанный JWT передаётся в заголовке `Authorization` запроса.

3. Push-сервис проверяет подпись, используя сохранённый публичный ключ.

```
POST https://fcm.googleapis.com/... HTTP/1.1
Authorization: vapid t=eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiJ9..., k=...
Content-Type: application/octet-stream
Content-Encoding: aes128gcm
```

Шифрование полезной нагрузки

Сообщения, передаваемые через push-сервис, **обязательно шифруются**. Это гарантирует, что даже push-сервис (который может принадлежать третьей стороне, например Google) не сможет прочитать содержимое уведомления.
Долгое время стандартом шифрования был **RFC 8291**, использующий ECDH (эллиптическую криптографию) в сочетании с AES-128-GCM.

Переход на HPKE

В июле 2026 года опубликован новый Internet-Draft **«WebPush Encryption using HPKE»** (Hybrid Public Key Encryption), который **заменяет RFC 8291**.
Причина перехода — **подготовка к пост-квантовой эпохе**: текущая эллиптическая криптография уязвима перед квантовыми компьютерами. HPKE использует гибридную схему шифрования, сочетающую механизм инкапсуляции ключей (KEM) и предварительно разделённый ключ (PSK) с аутентифицированным шифрованием с дополнительными данными (AEAD).

> **Важно**: на сентябрь 2026 года HPKE-документ остаётся **Internet-Draft** (рабочим черновиком) со сроком действия до 25 января 2027 года. Он ещё не является утверждённым RFC, но является официальным направлением развития стандарта.

Параллельно разрабатывается альтернативный подход — **«WebPush Encryption using Symmetric Ciphers»**, определяющий использование чисто симметричной криптографии с Web Push.

Жизненный цикл push-подписки

Создание подписки

```
// В основном потоке
async function subscribeUser() {
  const registration = await navigator.serviceWorker.ready;
  const subscription = await registration.pushManager.subscribe({
    userVisibleOnly: true,
    applicationServerKey: urlBase64ToUint8Array(publicVapidKey)
  });

  // Отправляем subscription на сервер
  await fetch('/api/subscribe', {
    method: 'POST',
```

```
    body: JSON.stringify(subscription),
    headers: { 'Content-Type': 'application/json' }
  });
}
```

Объект `PushSubscription` содержит:
- `endpoint` — URL для отправки сообщений
- `keys` — объект с публичными ключами (`p256dh` — ключ для шифрования, `auth` — секрет для аутентификации)

Важно: `userVisibleOnly: true` означает, что каждое push-сообщение должно сопровождаться видимым уведомлением — браузеры не разрешают «тихие» push-сообщения без отображения нотификации.

Хранение подписки на сервере

Подписку необходимо сохранять на сервере, привязав к конкретному пользователю. Без этого вы не сможете отправить сообщение — endpoint является единственным способом достичь браузера пользователя.

```
// Пример хранения в базе данных
const subscription = {
  endpoint: 'https://fcm.googleapis.com/...',
  expirationTime: null,
  keys: {
    p256dh: 'BP...',
    auth: 'C...'
  }
};
```

Отправка сообщения

На сервере отправка выглядит так (пример с библиотекой `web-push` для Node.js):

```
const webpush = require('web-push');
webpush.setVapidDetails(
  'mailto:example@yourdomain.com',
  publicVapidKey,
  privateVapidKey
);

await webpush.sendNotification(
  subscription, // сохранённая подписка
  JSON.stringify({ title: 'Hello!', body: 'World' })
);
```

Альтернативные реализации доступны для различных экосистем: Go (`go-webpush`), PHP, .NET, Deno и другие.

Declarative Web Push — новый стандарт

Традиционный Web Push требует, чтобы service worker выполнял JavaScript при каждом получении сообщения. Это создаёт несколько проблем:
- Каждое push-событие требует запуска JS-кода
- Каждое «тихое» push-сообщение расходует бюджет браузера
- Доставка зависит от исправности service worker

Declarative Web Push решает эти проблемы. Вместо выполнения кода сервер отправляет небольшой JSON-документ с описанием уведомления, и браузер рендерит его напрямую.

Статус стандарта

Declarative Web Push перешёл из статуса Safari-эксперимента в **W3C Working Draft**. W3C Push API Working Draft был опубликован 1 декабря 2025 года. Стандарт получил поддержку нескольких вендоров (multi-vendor editorship).

Формат полезной нагрузки:

```
{
  "web_push": 8030,
  "notification": {
    "title": "Your order has shipped",
    "body": "Tracking number AT-48201 is on the way.",
    "navigate": "https://example.com/orders/48201",
    "icon": "https://example.com/icon.png",
    "app_badge": 3
  }
}
```

Ключ `"web_push": 8030` (ссылка на RFC 8030) сигнализирует браузеру, что это декларативное уведомление. Поле `navigate` определяет URL, который откроется при клике — без необходимости писать `notificationclick`-обработчик. Поле `app_badge` обновляет бейдж на иконке приложения на домашнем экране.

Поддержка браузеров:

Платформа	Версия	Дата	Детали
iOS / iPadOS	18.4	Март 2025	Первый публичный релиз для Home Screen web apps
macOS / Safari	18.5	Май 2025	Поддержка на Mac для установленных и обычных сайтов
Safari	26.0 – 26.4	Сентябрь 2025 – Март 2026	Улучшение developer ergonomics, автоматическая пауза service worker в Web Inspector
iOS 26	—	2026	Все сайты, добавленные на Home Screen, по умолчанию открываются как web app

Apple выступила лидером внедрения. Mozilla заняла позитивную позицию в отношении стандарта. К 2026 году декларативный формат стал предпочтительным для iOS и macOS веб-пушей.

Ограничения и квоты

Chrome Push API rate limits

С января 2026 года Chrome начал внедрять **ограничения скорости** для Push API на основе вовлечения пользователей. Это часть долгосрочной стратегии Google по борьбе со спам-уведомлениями.
Как работает:
Ежедневно рассчитываются три ключевых фактора:
1. Количество push-сообщений на единицу времени, проведённого на сайте
2. Количество показанных запросов разрешения на единицу времени
3. Уровень вовлечения пользователя с сайтом (на основе engagement score и времени в фокусе)

Когда сайт признаётся **нарушающим** (disruptive) — отправляет большое количество уведомлений при очень низком вовлечении — его способность отправлять сообщения ограничивается до **не менее 1000 сообщений в минуту**.

Дополнительные меры Chrome:
- Автоматическое удаление разрешений на уведомления для сайтов, с которыми пользователь не взаимодействовал недавно
- Автоматический отзыв разрешений для сайтов, помеченных Google Safe Browsing как злоупотребляющие
- On-device машинное обучение для выявления спам-уведомлений на Android

Надёжность доставки

Push-доставка никогда не гарантируется на 100%. Причины:

- Браузер не запущен на устройстве
- Устройство офлайн
- Подписка истекла (токены могут «протухать» без предупреждения)
- Пользователь отозвал разрешение

Важное ограничение: Firefox не может отправлять уведомления на неактивные сайты, что ограничивает охват по сравнению с Chrome и Edge.
Рекомендуется реализовать:
- **Retry-механизмы** с экспоненциальной задержкой
- **Мониторинг** истечения подписок (поле `expirationTime` в `PushSubscription`)
- Обработку ошибок 410 (Gone) — сигнал, что подписка более не действительна

6. Поддержка браузеров

Браузер	Windows	macOS	Android	iOS
Chrome	✔	✔	✔	✔
Firefox	✔	✔	✔	✔
Safari	✖	✔ (macOS 13+)	✖	✔ (iOS 16.4+)
Edge	✔	✔	✔	✔
Opera	✔	—	✔	—

Ключевые требования:

- **HTTPS** — обязателен для всех браузеров
- **Явное согласие пользователя** — запрос разрешения должен инициироваться действием пользователя

Особенности Safari:

- Использует Apple Push Notification Service (APNS)
- iOS 17.4: пользователи в Евросоюзе больше не могут устанавливать PWA на Home Screen, что блокирует уведомления Safari для этого региона

Безопасность и конфиденциальность

Разрешения пользователя

Браузеры **НЕ ДОЛЖНЫ** предоставлять доступ к Push API без явного согласия пользователя. Это означает, что:

- Запрос разрешения должен инициироваться действием пользователя (клик, тап)
- Пользователь может в любой момент отозвать разрешение в настройках браузера

Конфиденциальность endpoint'ов

Push-эндпоинты **НЕ ДОЛЖНЫ** раскрывать информацию о пользователе — устройство, идентичность или местоположение. Push-сервис не должен позволять коррелировать разные подписки одного пользователя.

Метаданные vs содержимое

Хотя содержимое сообщения зашифровано, push-сервис всё равно видит **метаданные**: факт отправки, время, размер сообщения, частоту. Это неизбежный компромисс: push-сервис должен знать, кому и когда доставлять сообщение.

CORS

В сообществе обсуждается необходимость явной поддержки **CORS** для push-серверов, чтобы упростить одноранговые (P2P) сценарии.

Практические рекомендации

UX получения разрешения

Никогда не запрашивайте разрешение на push-уведомления сразу при загрузке страницы. Пользователь должен понимать, зачем ему это нужно. Рекомендуется:

1. Показать поясняющий UI («Хотите получать уведомления о новых сообщениях?»)
2. Запросить разрешение только после явного согласия пользователя
3. Обработать отказ — не блокировать функциональность

Управление подписками на сервере

Храните для каждого пользователя:

- Объект `PushSubscription`
- Дату создания подписки
- Признак активности (если пришла ошибка 410 — удаляйте)

Реализуйте эндпоинты:

- `POST /api/push/subscribe` — сохранение подписки
- `DELETE /api/push/unsubscribe` — удаление при отписке

Оптимизация полезной нагрузки

- Ограничьте размер сообщения — стандартный лимит: до 70 символов для заголовка, 250 для тела
- Используйте сжатие (Content-Encoding: aes128gcm уже включает сжатие)
- Для сложных данных используйте ID ссылки вместо полного объекта

Мониторинг и отладка

- В Chrome DevTools → Application → Service Workers можно симулировать push-события
- Safari 26 добавил автоматическую паузу service worker в Web Inspector для отладки push-событий
- Логируйте ошибки отправки на сервере для выявления проблемных подписок

Генерация VAPID-ключей

```
# Пример генерации с использованием web-push CLI
npx web-push generate-vapid-keys
```

Или программно:

```
keys, err := webpush.GenerateVAPIDKeys()
pem, err := keys.ExportVAPIDPrivateKeyPEM()
```

Web Push vs нативные push-уведомления

Характеристика	Web Push	Нативные Push
Требования	HTTPS, Service Worker	SDK, сертификаты
Доставка	Зависит от браузера	Управляется ОС
Срок жизни токена	Может истечь без предупреждения	Стабильнее
Кастомизация	Ограничена API браузера	Полный контроль
Стоимость	Бесплатно	Требует аккаунтов разработчика

Web Push идеально подходит для:

- PWA и веб-приложений
- Быстрого прототипирования
- Сценариев, где нативное приложение нецелесообразно

Будущее Web Push

Экосистема Web Push активно развивается. Ключевые тренды на 2026 год:

1. **HPKE вместо RFC 8291** — Internet-Draft от июля 2026 года, переход на пост-квантово-устойчивое шифрование
2. **Declarative Web Push** — стандартизация в W3C, расширение поддержки в браузерах
3. **Ужесточение политик** — Chrome вводит лимиты на частоту отправки для сайтов с низким вовлечением
4. **Унификация платформ** — iOS 16.4+ добавил поддержку Web Push, существенно расширив охват
5. **Более широкий PWA-охват** — iOS 26: все сайты на Home Screen по умолчанию открываются как web app

Заключение

Web Push — это мощный, но сложный механизм, требующий понимания нескольких уровней абстракции: от service worker'ов в браузере до криптографических протоколов на сервере. Мы разобрали архитектуру, протоколы VAPID и шифрования (включая переход на HPKE), процесс подписки и отправки, а также новые стандарты, такие как Declarative Web Push.

Ключевые выводы:

- **Service worker** — обязательный компонент для приёма сообщений (кроме Declarative Web Push)
- **VAPID** — аутентификация сервера приложений (RFC 8292)
- **Шифрование** — обязательное условие (переход с RFC 8291 на HPKE)
- **Endpoint'ы** — чувствительные данные, требуют безопасного хранения
- **Declarative Web Push** — будущее стандарта (W3C Working Draft), упрощающее реализацию
- **Rate limits** — Chrome внедряет ограничения на основе вовлечения пользователей с 2026 года

При правильной реализации Web Push становится надёжным каналом коммуникации с пользователями, повышая вовлечённость и ценность веб-приложений.