GUÍA PASO A PASO

Alertas de checkout para el Buen Fin

Un script en Node, sin dependencias, que cada 10 minutos compara tus pedidos con la semana pasada, cuenta pagos fallidos y checkouts abandonados y prueba que el checkout cargue, con aviso por Slack y por correo. Probado con respuestas simuladas, con la checklist para el Buen Fin.

PARA	Tiendas Shopify que quieren saber en minutos si el checkout deja de vender
PROBADO CON	Node 22.22.3 · node:test con fetch simulado · consultas validadas con el validador de Shopify (2026-10) · Windows 11
ACTUALIZADA	11 de noviembre de 2026
DOCUMENTACIÓN	shopify.dev/docs/api/admin-graphql/2026-10
ARTÍCULO	ecomlabs.dev/g/monitoreo
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	El script
Paso 02	Prográmalo y pruébalo
Al terminar	Checklist del Buen Fin

ANTES DE EMPEZAR

Qué vas a montar

`monitor.mjs` corre cada 10 minutos en GitHub Actions y revisa tres alertas: pedidos de la última hora contra la misma hora de la semana pasada, errores de pago (transacciones fallidas y proporción de checkouts abandonados) y una prueba del checkout. Si algo falla, avisa por Slack y la ejecución queda en rojo, lo que dispara el correo de GitHub. Node 18 o superior, sin dependencias, API 2026-10.

Crea una app en el Dev Dashboard con el permiso `read_orders` e instálala en tu tienda. Guarda en los secretos de Actions `SHOP` (`tu-tienda.myshopify.com`), `SHOPIFY_CLIENT_ID`, `SHOPIFY_CLIENT_SECRET`, `PROBE_VARIANT_ID` (`gid://shopify/ProductVariant/…`) y `SLACK_WEBHOOK_URL`. Con una app personalizada anterior a 2026, usa su token como `SHOPIFY_ADMIN_TOKEN` y agrégalo al `env` del workflow. Guarda el script como `monitor.mjs` en la raíz del repositorio; sus umbrales son un ejemplo ilustrativo.

PASO 01

El script

```
// monitor.mjs: three checkout alerts for a Shopify store. Node 18+, no dependencies.
import { fileURLToPath } from "node:url";

const API_VERSION = "2026-10";
const { SHOP, SHOPIFY_CLIENT_ID, SHOPIFY_CLIENT_SECRET, PROBE_VARIANT_ID, SLACK_WEBHOOK_URL } = process.env;
const { SHOPIFY_ADMIN_TOKEN } = process.env; // only if you already have a legacy custom app token
const DROP_RATIO = 0.5; // alert 1: fewer than half of last week's orders...
const MIN_BASELINE = 10; // ...when last week had at least 10 in that hour
const MAX_FAILED = 3; // alert 2: failed or errored transactions in the hour
const ABANDON_JUMP = 0.2; // alert 2: abandoned share up 20 points vs last week
const MINUTE = 60_000;
const HOUR = 60 * MINUTE;
const WEEK = 7 * 24 * HOUR;

const COUNTS = `query Counts($now: String!, $lastWeek: String!) {
  ordersNow: ordersCount(query: $now) { count }
  ordersLastWeek: ordersCount(query: $lastWeek) { count }
  abandonedNow: abandonedCheckoutsCount(query: $now) { count }
  abandonedLastWeek: abandonedCheckoutsCount(query: $lastWeek) { count }
}`;
const RECENT_ORDERS = `query RecentOrders($q: String!) {
  orders(first: 50, query: $q, sortKey: CREATED_AT, reverse: true) {
    nodes { name transactions(first: 5) { status kind gateway errorCode } }
  }
}`;
const PROBE = `mutation Probe($variantId: ID!) {
  cartCreate(input: { lines: [{ merchandiseId: $variantId, quantity: 1 }] }) {
    cart { checkoutUrl totalQuantity }
    userErrors { message }
  }
```

```
}`;

const between = (end) =>
  `created_at:>='${new Date(end - HOUR).toISOString()}' created_at:<'${new Date(end).toISOString()}'`;
const postJson = (url, payload, headers = {}) =>
  fetch(url, { method: "POST", headers: { "Content-Type": "application/json", ...headers }, body:
JSON.stringify(payload) });

async function getToken() {
  if (SHOPIFY_ADMIN_TOKEN) return SHOPIFY_ADMIN_TOKEN;
  const res = await fetch(`https://${SHOP}/admin/oauth/access_token`, {
    method: "POST",
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
    body: new URLSearchParams({
      grant_type: "client_credentials",
      client_id: SHOPIFY_CLIENT_ID,
      client_secret: SHOPIFY_CLIENT_SECRET,
    }),
  });
  if (!res.ok) throw new Error(`token request: HTTP ${res.status}`);
  return (await res.json()).access_token;
}

async function admin(token, query, variables) {
  const url = `https://${SHOP}/admin/api/${API_VERSION}/graphql.json`;
  const res = await postJson(url, { query, variables }, { "X-Shopify-Access-Token": token });
  const body = await res.json().catch(() => ({}));
  if (!res.ok || body.errors) throw new Error(`HTTP ${res.status} ${JSON.stringify(body.errors ?? "")}`);
  return body.data;
}

// Alert 1: orders in the hour vs the same hour last week.
function ordersAlert({ ordersNow, ordersLastWeek }) {
  if (ordersLastWeek.count >= MIN_BASELINE && ordersNow.count < ordersLastWeek.count * DROP_RATIO) {
    return `Orders: ${ordersNow.count} in the hour vs ${ordersLastWeek.count} same hour last week`;
  }
}

// Alert 2: failed transactions on recent orders, and abandoned checkouts (where declines land).
async function paymentAlerts(token, q, c) {
  const alerts = [];
  // shortcut: reads the latest 50 orders of the hour; page with `after` if you sell more per hour.
  const { orders } = await admin(token, RECENT_ORDERS, { q });
  const failed = orders.nodes.flatMap((o) =>
    o.transactions
      .filter((t) => t.status === "FAILURE" || t.status === "ERROR")
      .map((t) => `${o.name} ${t.kind} ${t.gateway} ${t.errorCode ?? t.status}`),
  );
  if (failed.length >= MAX_FAILED) alerts.push(`Payments: ${failed.length} failed (${failed.slice(0,
3).join("; ")})`);
  const share = (abandoned, orders) => (abandoned + orders ? abandoned / (abandoned + orders) : 0);
```

```
  const now = share(c.abandonedNow.count, c.ordersNow.count);
  const before = share(c.abandonedLastWeek.count, c.ordersLastWeek.count);
  if (c.abandonedNow.count >= MIN_BASELINE && now - before > ABANDON_JUMP) {
    alerts.push(`Abandoned checkouts: ${Math.round(now * 100)}% vs ${Math.round(before * 100)}% last week`);
  }
  return alerts;
}

// Alert 3: a synthetic check. Creates a cart (never an order) and loads its checkout URL.
async function probeCheckout() {
  const variables = { variantId: PROBE_VARIANT_ID };
  const res = await postJson(`https://${SHOP}/api/${API_VERSION}/graphql.json`, { query: PROBE, variables });
  const body = await res.json().catch(() => ({}));
  const cart = body.data?.cartCreate?.cart;
  if (!res.ok || !cart?.totalQuantity) {
    return `cartCreate: HTTP ${res.status} ${JSON.stringify(body.errors ?? body.data?.cartCreate?.userErrors
?? "")}`;
  }
  const page = await fetch(cart.checkoutUrl);
  if (!page.ok) return `checkout page: HTTP ${page.status}`;
}

async function checkoutAlert(sleep) {
  const first = await probeCheckout().catch((err) => err.message);
  if (!first) return;
  await sleep(30_000); // one retry, so a single blip does not page anyone
  const second = await probeCheckout().catch((err) => err.message);
  if (second) return `Checkout probe failed twice: ${second}`;
}

export async function runChecks({ now = Date.now(), sleep = (ms) => new Promise((r) => setTimeout(r, ms)) } =
{}) {
  const alerts = [];
  const end = now - 15 * MINUTE; // a checkout counts as abandoned after 10+ minutes
  try {
    const token = await getToken();
    const counts = await admin(token, COUNTS, { now: between(end), lastWeek: between(end - WEEK) });
    alerts.push(ordersAlert(counts), ...(await paymentAlerts(token, between(end), counts)));
  } catch (err) {
    alerts.push(`Monitor could not read the Admin API: ${err.message}`);
  }
  alerts.push(await checkoutAlert(sleep));
  return alerts.filter(Boolean);
}

export async function main() {
  const alerts = await runChecks();
  if (!alerts.length) return console.log("ok");
  const text = `Checkout alert for ${SHOP}\n- ${alerts.join("\n- ")}`;
  console.log(text);
  const res = SLACK_WEBHOOK_URL && (await postJson(SLACK_WEBHOOK_URL, { text }));
```

```
  if (res && !res.ok) console.error(`Slack: HTTP ${res.status} ${await res.text()}`);
  process.exitCode = 1; // a failed run also emails you through GitHub Actions
}

if (process.argv[1] === fileURLToPath(import.meta.url)) await main();
```

PASO 02

Prográmalo y pruébalo

```
# .github/workflows/checkout-monitor.yml
name: Checkout monitor
on:
  schedule:
    - cron: "5,15,25,35,45,55 * * * *"
  workflow_dispatch:
jobs:
  monitor:
    runs-on: ubuntu-latest
    timeout-minutes: 5
    steps:
      - uses: actions/checkout@v7
      - uses: actions/setup-node@v7
        with:
          node-version: 22
      - run: node monitor.mjs
        env:
          SHOP: ${{ secrets.SHOP }}
          SHOPIFY_CLIENT_ID: ${{ secrets.SHOPIFY_CLIENT_ID }}
          SHOPIFY_CLIENT_SECRET: ${{ secrets.SHOPIFY_CLIENT_SECRET }}
          PROBE_VARIANT_ID: ${{ secrets.PROBE_VARIANT_ID }}
          SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
```

El `cron` va en UTC y evita el minuto 0, cuando GitHub puede retrasar o descartar ejecuciones. Activa las notificaciones de Actions. En tu tienda, corre el workflow a mano (verde, `ok`); luego pon un `PROBE_VARIANT_ID` que no exista y córrelo (Slack, rojo, correo); regresa el secreto.

AL TERMINAR

Checklist del Buen Fin

- [ ] Antes del jueves 12: una ejecución manual en verde y una alerta forzada que llegó a Slack y al correo.
- [ ] La variante de prueba tiene inventario para toda la semana.
- [ ] Del 13 al 17 de noviembre, alguien de guardia recibe Slack y revisa Actions; después, desactiva el workflow.
- [ ] Si salta la alerta 2, lee la línea de tiempo de los checkouts abandonados: la API no da el motivo.