EcomLabs

STEP-BY-STEP GUIDE

Checkout alerts for big sale days

A Node script with no dependencies that runs every 10 minutes, compares your orders with last week, counts failed payments and abandoned checkouts and checks that the checkout loads, then alerts you on Slack and by email. Tested against stubbed responses, with a Buen Fin checklist.

FOR	Shopify stores that want to know within minutes when the checkout stops selling
TESTED WITH	Node 22.22.3 · node:test with a stubbed fetch · queries checked with Shopify's validator (2026-10) · Windows 11
UPDATED	November 11, 2026
DOCS	shopify.dev/docs/api/admin-graphql/2026-10
ARTICLE	ecomlabs.dev/g/monitoreo-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will set up
Step 01	The script
Step 02	Schedule it and test it
When you finish	Buen Fin checklist

BEFORE YOU START

What you will set up

`monitor.mjs` runs every 10 minutes on GitHub Actions and checks three alerts: orders in the last hour against the same hour last week, payment errors (failed transactions and the share of abandoned checkouts) and a checkout probe. When something fails, it posts to Slack and the run turns red, which triggers GitHub's email. Node 18 or later, no dependencies, API 2026-10.

Create an app in the Dev Dashboard with the `read_orders` scope and install it on your store. Store `SHOP` (`your-store.myshopify.com`), `SHOPIFY_CLIENT_ID`, `SHOPIFY_CLIENT_SECRET`, `PROBE_VARIANT_ID` (`gid://shopify/ProductVariant/…`) and `SLACK_WEBHOOK_URL` as Actions secrets. With a custom app created before 2026, use its token as `SHOPIFY_ADMIN_TOKEN` and add it to the workflow's `env`. Save the script as `monitor.mjs` at the root of the repository; its thresholds are an illustrative example.

STEP 01

The 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();
```

STEP 02

Schedule it and test it

```
# .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 }}
```

The `cron` runs in UTC and skips minute 0, when GitHub can delay or drop scheduled runs. Turn on Actions notifications. On your store, run the workflow by hand (green, `ok`); then set a `PROBE_VARIANT_ID` that does not exist and run it (Slack, red, email); put the secret back.

WHEN YOU FINISH

Buen Fin checklist

- [ ] The day before the sale: a manual run ends green and a forced alert reached Slack and your email.
- [ ] The probe variant has stock for the whole week.
- [ ] November 13 to 17: someone on call gets Slack and checks Actions; afterward, disable the workflow.
- [ ] If alert 2 fires, read the timeline of the abandoned checkouts: the API gives no reason.