KARMA ADMIN CONSOLE

Dashboard Bundle Developer Guide

How to build a self-contained static dashboard that passes validation, renders correctly in the console, and stays fast.

50 MB / 300 files hard cap | < 2 MB target | No network at runtime | Sandboxed, null origin

Version **1.0** Applies to **Dashboard Hosting v1** Audience **Dashboard authors**

1 What a hosted dashboard is

A folder of plain static files — HTML, CSS, JavaScript — with the data already inside it, zipped and uploaded.

✓ IT IS
A self-contained static site. Data baked in at export time. Charts drawn from that data in the browser. Everything it needs travels inside the zip.

✕ IT IS NOT
A web app. There is no backend, no database, no login, no live data feed, and no way to call an API — not even yours.

Think “an interactive PDF that can do maths”, not “a web application”. The dashboard shows a **snapshot**. To refresh the numbers you export a new bundle and upload it as a new version.

> **Why it works this way.** Bundles are authored outside the console and run inside it. Isolating them completely — no network, no storage, no access to the host page — is what makes it safe to host arbitrary uploaded code next to member data.

2 Size: aim small

IDEAL
< 2 MB
Pure HTML/CSS/JS. Data as inline JSON or JS. Charts drawn in SVG or canvas. Loads instantly.

ACCEPTABLE
< 10 MB
A few optimised images, one bundled font family. Noticeably slower on a poor connection.

HARD CEILING
50 MB
Also max 300 files. Rejected beyond this. If you are near it, something has gone wrong.

The whole bundle is fetched before anything renders, so weight is felt directly by whoever opens it — often an executive on a phone. A well-built dashboard is usually **under 500 KB**.

BLOAT CAUSE	TYPICAL COST	FIX
Chart exported as PNG/JPG	200–800 KB each	Draw it in SVG from the data. A bar chart is a few hundred *bytes*, and it scales.
Uncompressed screenshots	1–5 MB each	Convert to `.webp`, resize to the size actually displayed.
Embedded video	10 MB+	Not possible — video extensions are not allowed. Link out, or use a still.
Base64 blobs in HTML	+33% over the file	Ship the file separately and reference it relatively.
Whole charting library	300 KB–1 MB	Hand-rolled SVG covers bars, lines and numbers. Bundle a library only if truly needed — one copy.
Full webfont family	400 KB–2 MB	Use system fonts. If brand type is essential, one `.woff2` weight.

3 Bundle structure

```
manifest.json ← required, at the ROOT of the zip
index.html ← entry point (or whatever manifest.entry names)
styles.css
app.js
data.js ← your data, baked in
assets/logo.svg ← subfolders are fine
```

manifest.json

```
{
  "name": "Quarterly Revenue",
  "version": "1.0.0",
  "entry": "index.html",
  "description": "Revenue and occupancy for the last six months."
}
```

`name`, `version` and `entry` are required strings. `description` is optional. If the manifest is missing or malformed, the platform falls back to `index.html` at the root — and rejects the upload if that isn't there either.

Allowed file types

Exactly these extensions. Anything else is rejected:

`html` `css` `js` `json` `png` `jpg` `jpeg` `gif` `svg` `webp` `woff` `woff2` `ttf` `otf` `ico`

> **Watch out for:** `.map` source maps (turn them off), `README.md`, `.DS_Store`, `.gitignore`, and any extensionless file. All are rejected. Clean the folder before zipping.

4 What gets your upload rejected

These are checked server-side before anything is stored. The error text below is what you will actually see.

PROBLEM	ERROR YOU'LL SEE	FIX
Disallowed extension	*Archive contains N file(s) with disallowed extensions: …*	Remove the file or convert it. Every offender is listed at once.
Too many files	*Archive contains N files, which exceeds the limit of 300.*	Consolidate assets; drop unused ones.
Too large	*Archive declares N uncompressed bytes, which exceeds the limit of 52428800.*	See §2. Usually images.
No entry point	*Could not determine an entry point: no manifest.json entry and no index.html at root.*	Almost always a wrapper folder — see §9.
Path traversal	*Archive contains N entries with unsafe paths (absolute, or escaping the bundle root): …*	No `../`, no leading `/`, no drive letters, no backslashes.
Symlinks	*Archive contains symlink entries, which are not allowed: …*	Zip real files. Use “compress” rather than a tar-style archiver that preserves links.
Duplicate paths	*Archive contains duplicate paths: …*	Two entries resolved to the same path; rebuild the zip.
Corrupt zip	*Upload is not a readable zip archive: …*	Re-zip. Don't rename a `.rar` / `.7z` to `.zip`.

> Validation is **all-or-nothing**. A rejected upload stores nothing at all, so you can fix and retry immediately without leaving debris behind.

5 The runtime environment

This is the part that surprises people. Your dashboard runs in a locked sandbox, and the restrictions are **not** obvious from the code.

It renders in an iframe with `sandbox="allow-scripts"` and no `allow-same-origin` — a **null origin** — under this policy:

```
Content-Security-Policy:
  default-src 'none'; script-src 'unsafe-inline' 'self'; style-src 'unsafe-inline' 'self';
  img-src 'self' data:; font-src 'self' data:; connect-src 'none'; form-action 'none';
  base-uri 'none'; frame-ancestors 'self'
```

✕ NETWORK CALLS — silently blocked

```
fetch('/api/data')
new WebSocket(…)
navigator.sendBeacon(…)
```

✓ BAKE THE DATA IN

```
<script src="data.js"></script>
window.DASHBOARD_DATA = {…}
```

✕ STORAGE — throws SecurityError

```
localStorage.setItem(…)
sessionStorage / document.cookie
```

✓ PLAIN VARIABLES

```
let state = { tab: 'revenue' };
```

State lasts for the page view. That's enough.

✕ CDN & EXTERNAL FONTS

```
<script src="https://cdn…/chart.js">
<link href="https://fonts.googleapis…">
```

✓ BUNDLE IT, REFERENCE RELATIVELY

```
<script src="vendor/chart.min.js">
@font-face { src: url("fonts/x.woff2") }
```

✕ EXTERNAL IMAGES

```
<img src="https://example.com/logo.png">
```

✓ LOCAL FILE OR DATA URI

```
<img src="assets/logo.webp">
```

✕ HOST ACCESS & NAVIGATION

```
window.parent.… / window.top.…
<form action="…"> / window.open(…)
```

✓ SELF-CONTAINED INTERACTION

```
button.addEventListener('click', render)
```

What does work: inline `<script>` and `<style>`, inline event handlers, `data:` URIs for images and fonts, and every ordinary DOM, SVG, canvas and JS API that doesn't touch the network or storage. `eval()` and `new Function()` are blocked.

> **Failures are quiet.** A blocked `fetch` rejects a promise nobody catches; the dashboard just renders empty. Test with the network disconnected — if it still works, it will work here.

6 Build responsively

It renders in an iframe that fills the console's viewport, from roughly **360 px** wide upward — and share-link recipients often open it on a phone.

- Include `<meta name="viewport" content="width=device-width, initial-scale=1">`.
- Use fluid layout: `max-width` containers, CSS grid with `repeat(auto-fit, minmax(…, 1fr))`, `clamp()` for type.
- Never fix the page shell to a pixel width, and don't rely on hover — it may be viewed on touch.
- Give SVG charts a `viewBox` and `width:100%; height:auto` so they scale.
- Optional but appreciated: honour `prefers-color-scheme`; the console has light and dark modes.

7 A complete working example

Five files, 4,122 bytes total — verified against the real validator. Copy this as a starting point.

manifest.json

```
{
  "name": "Quarterly Revenue",
  "version": "1.0.0",
  "entry": "index.html",
  "description": "Revenue and occupancy for the last six months."
}
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Quarterly Revenue</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <main class="wrap">
      <header>
        <h1>Quarterly Revenue</h1>
        <p class="sub">Six months to March 2026 &middot; all resorts</p>
      </header>
      <section class="tiles" id="tiles"></section>
      <section class="card">
        <h2>Revenue by month</h2>
        <div id="chart" class="chart"></div>
      </section>
      <footer class="foot">Data baked in at export &middot; generated 12 Mar 2026</footer>
    </main>

    <!-- Data lives IN the bundle. No fetch, no API, no CDN. -->
    <script src="data.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

data.js

```
/* Baked-in data. Replace at export time; never fetched at runtime. */
window.DASHBOARD_DATA = {
  totals: [
    { key: "Revenue", value: "$4.82M" },
    { key: "Occupancy", value: "78.4%" },
    { key: "ADR", value: "$212" },
    { key: "Bookings", value: "9,431" }
  ],
  months: [
    { label: "Oct", value: 690000 }, { label: "Nov", value: 745000 },
    { label: "Dec", value: 910000 }, { label: "Jan", value: 820000 },
    { label: "Feb", value: 788000 }, { label: "Mar", value: 867000 }
```

```
  ]
};
```

app.js

```
(function () {
  var d = window.DASHBOARD_DATA;

  document.getElementById("tiles").innerHTML = d.totals
    .map(function (t) {
      return '<div class="tile"><div class="k">' + t.key +
             '</div><div class="v">' + t.value + "</div></div>";
    }).join("");

  // Chart drawn as inline SVG — no library, no image, a few hundred bytes.
  var W = 720, H = 260, pad = 34;
  var max = Math.max.apply(null, d.months.map(function (m) { return m.value; }));
  var bw = (W - pad * 2) / d.months.length;

  var bars = d.months.map(function (m, i) {
    var h = Math.round(((H - pad * 2) * m.value) / max);
    var x = pad + i * bw + bw * 0.18, y = H - pad - h;
    return '<rect x="' + x + '" y="' + y + '" width="' + bw * 0.64 +
           '" height="' + h + '" rx="5" fill="var(--accent)" opacity="0.9"></rect>' +
           '<text x="' + (x + bw * 0.32) + '" y="' + (H - pad + 16) +
           '" fill="var(--dim)" font-size="11" text-anchor="middle">' + m.label + "</text>";
  }).join("");

  document.getElementById("chart").innerHTML =
    '<svg viewBox="0 0 ' + W + " " + H + '" role="img" aria-label="Revenue by month">' +
    '<line x1="' + pad + '" y1="' + (H - pad) + '" x2="' + (W - pad) +
    '" y2="' + (H - pad) + '" stroke="var(--line)"></line>' + bars + "</svg>";
})();
```

styles.css

```
/* Fluid layout, system fonts, both colour schemes. Abridged — see the skill file. */
:root { --bg:#0f1420; --panel:#171d2b; --line:#263047;
        --ink:#e8edf7; --dim:#94a3b8; --accent:#38bdf8; }
@media (prefers-color-scheme: light) {
  :root { --bg:#f6f8fc; --panel:#fff; --line:#e3e8f2; --ink:#0f172a; --dim:#5b6880; }
}
body { margin:0; background:var(--bg); color:var(--ink);
       font:15px/1.5 system-ui, -apple-system, "Segoe UI", Roboto, sans-serif; }
.wrap  { max-width:1100px; margin:0 auto; padding:clamp(16px,3vw,32px); }
.tiles { display:grid; gap:12px;
         grid-template-columns:repeat(auto-fit,minmax(150px,1fr)); }
.chart svg { display:block; width:100%; height:auto; }
```

8 Pre-upload checklist

- [ ] `manifest.json` is at the zip root
- [ ] `name`, `version`, `entry` all present
- [ ] The `entry` file exists and ends in `.html`
- [ ] Every extension is on the allowed list
- [ ] No `.map`, `.md`, `.DS_Store`
- [ ] Under 300 files
- [ ] Well under 50 MB — ideally under 2 MB
- [ ] No `fetch`/XHR/WebSocket anywhere
- [ ] No `localStorage`/`sessionStorage`/cookies
- [ ] No CDN, Google Fonts, or external images
- [ ] Every path relative — no leading `/`, no `../`
- [ ] Tested with the network disconnected
- [ ] Readable at 360 px wide
- [ ] Data snapshot date shown in the UI

9 Zipping and handing it over

> **The single most common mistake.** Zip the **contents** of your folder, not the folder itself. `manifest.json` and your entry HTML must sit at the **top level** of the archive. A wrapper folder is why uploads fail with *"no manifest.json entry and no index.html at root"*.

✕ WRONG — wrapper folder

```
bundle.zip
  my-dashboard/
    manifest.json
    index.html
```

✓ RIGHT — files at root

```
bundle.zip
  manifest.json
  index.html
  styles.css
```

macOS: open the folder, select all the files inside, right-click → *Compress N items*. **Windows:** open the folder, select all, right-click → *Send to* → *Compressed (zipped) folder*. In both cases you are selecting the *files*, never the enclosing folder.

Then send the `.zip` to a dashboard administrator, who uploads it via **Dashboards → Create Dashboard**. They'll choose who can see it and whether to issue a public share link. If validation fails, they will see the exact errors — ask for them, fix, and resend.

> **Updating later.** Send a new zip. Uploading it creates a new version; the previous one is kept and can be rolled back to instantly.