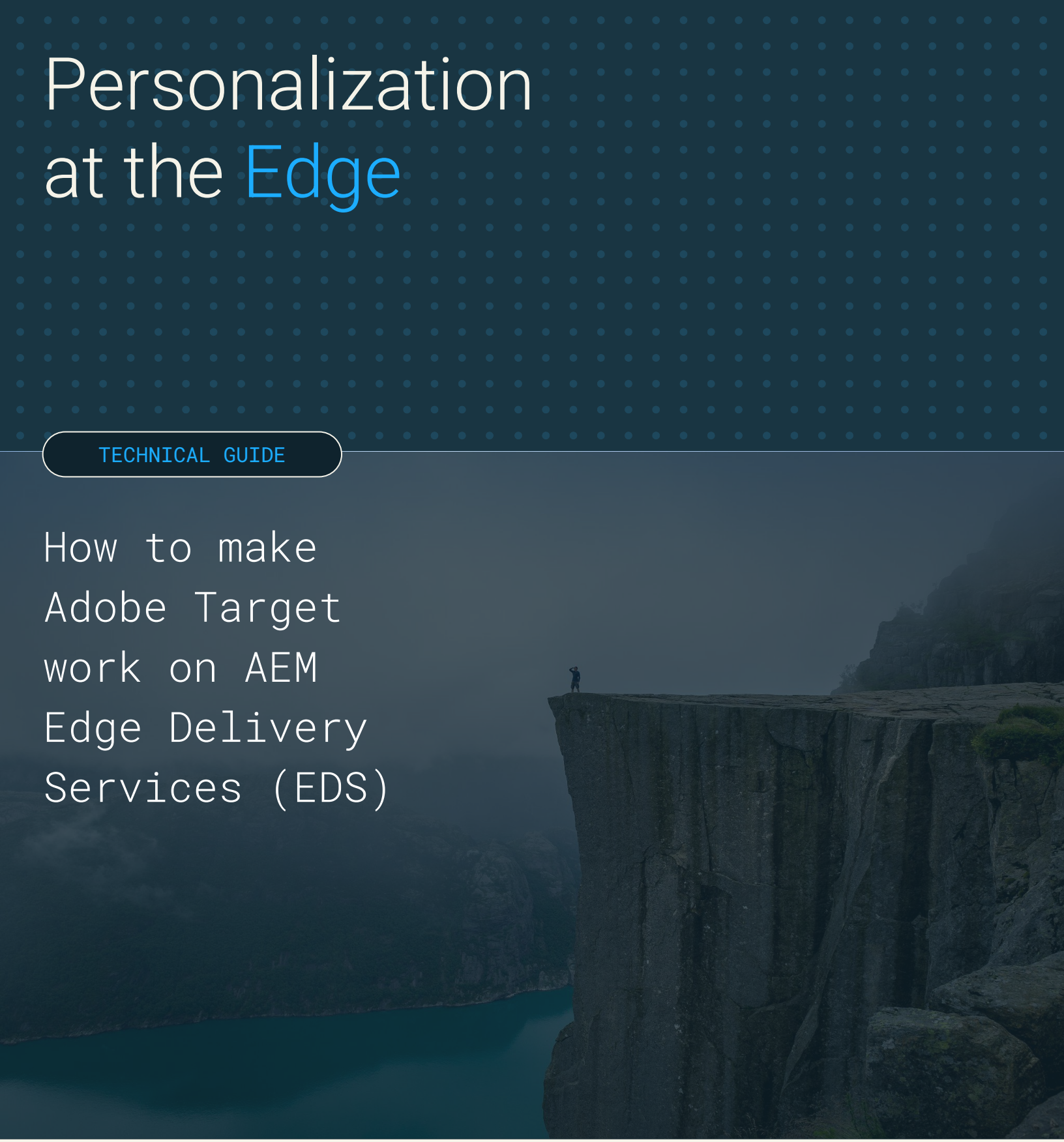
Personalization at the Edge

TECHNICAL GUIDE

How to make Adobe Target work on AEM Edge Delivery Services (EDS)



SHIFT PARADIGM

AUG • 2026

Email | Brand | Product | User Interface Design | User Experience | Customer Experience | Creative Direction | Full Stack Development | Business Strategy | MarTech

EXECUTIVE SUMMARY

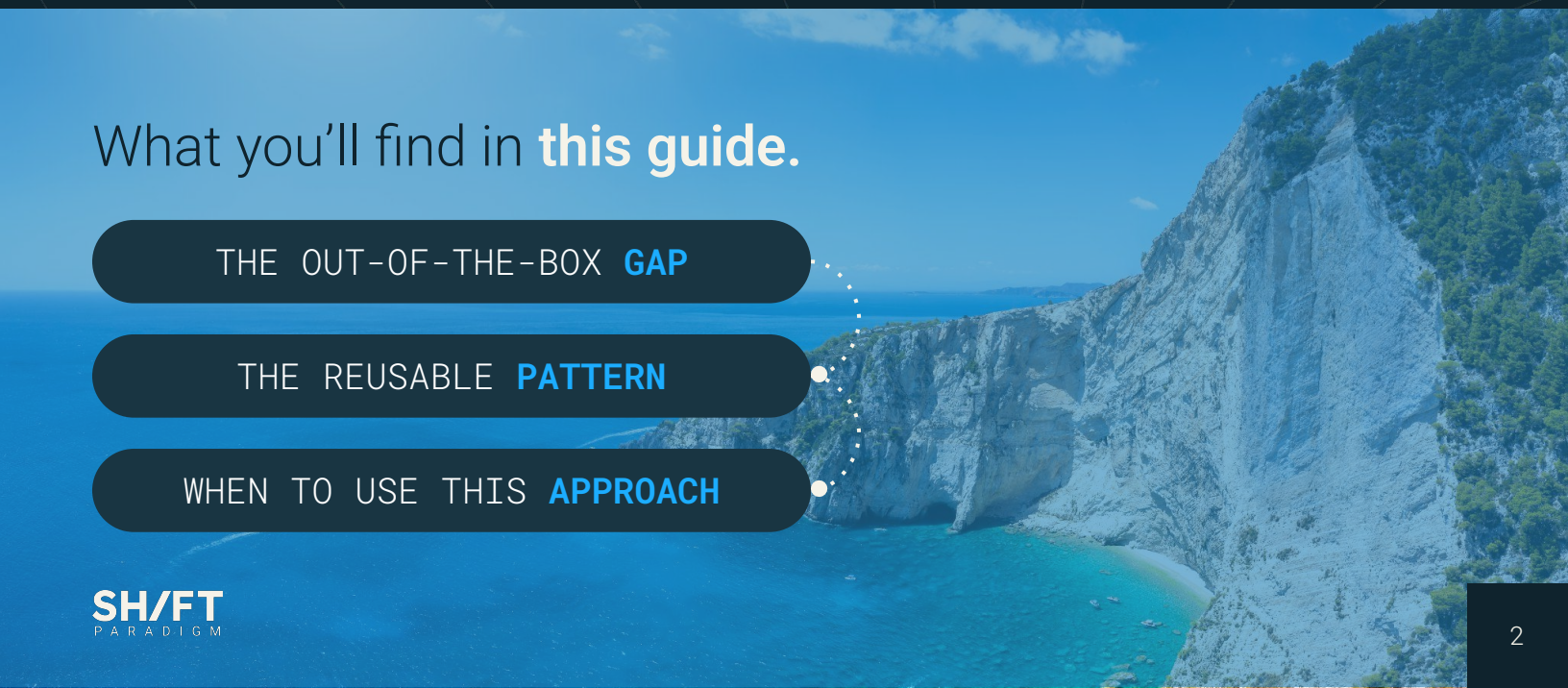
When out-of-the-box isn’t enough.

If you're standardized on the Adobe stack, there's a reasonable expectation that Adobe Target personalization on Experience Manager Edge Delivery Services (EDS) *should* just work out of the box.

But, the reality is more complicated.

EDS's performance-first architecture and Target's personalization model don't always align cleanly. It’s a gap we uncovered while modernizing an enterprise client's web experience, and know isn't unique to that one project.

This guide breaks down the pattern we engineered to close that gap—one we now deploy with confidence across both Web SDK and Document Authoring implementations.



What you’ll find in **this guide.**

- THE OUT-OF-THE-BOX **GAP**
- THE REUSABLE **PATTERN**
- WHEN TO USE THIS **APPROACH**

THE EXPECTATION

The promise of the “happy path.”

Adobe's Experience Platform Web SDK—the library most people know as `alloy.js`—ships with an elegantly simple model for personalization.

You send an event, ask it to render decisions, and the SDK drops the personalized content into the page for you.

```
JavaScript

-   const response = await window.alloy('sendEvent',
    { renderDecisions: true });

// Alloy renders into the DOM and auto-fires the impression
```



That one line does a lot of quiet work.

It applies to every decision, mitigates flicker, and fires the impression events Target needs for reporting.

And, it's a great developer experience *as long as one assumption holds*—the native SDK is built around the idea that when `sendEvent` returns, **the page DOM is complete and stable.**

Where Edge Delivery breaks the assumption.



Edge Delivery Services *doesn't ship a finished page.*

It ships a minimal HTML shell and then decorates blocks and sections progressively on the client, in two named lifecycle phases.

01 `loadEager`

runs *before* LCP (Largest Contentful Paint): only the page shell and the first section exist at this point

02 `loadLazy`

runs *after* LCP: the remaining sections and blocks decorate over the next several hundred milliseconds, each one signalling completion with two attributes

`data-section-status="loaded"`

`data-block-status="loaded"`

How this breaks the "happy path."

Target activities

Most Target activities are authored against decorated DOM

`.hero` `.cards block` `[data-block-name]` and a section wrapper

EDS

None of those elements exist when the SDK receives its personalization response

So, `renderDecisions: true` quietly misses its selectors, the page flickers, and section-level content gets double-wrapped

The happy path may be broken, *but* **the SDK isn't**—it's just answering a question EDS hasn't finished asking yet—and, this is exactly what makes EDS so fast.

So, we *changed* the contract.

Rather than fight the SDK, or constrain the client to a thin slice of what Target can do, we inverted the relationship. We call the Web SDK with `renderDecisions: false` and *own the apply phase ourselves.*

That custom layer isn't an optional workaround sitting on top of the integration, it's the **integration contract** between the Adobe Web SDK and AEM Edge Delivery. In practice, that means rebuilding several behaviors the SDK normally hands you for free.

How we do it: four key considerations in our code.

01 Apply in two waves, not one

02 Scope flicker control

03 Section-aware content handling

04 Manual impression firing everywhere



01 Apply in two waves, not one.

THE FIRST WAVE

Apply any target activity whose area on the page already exists at fetch time, *before the first section renders.*

```
if (adobeTarget) {
    const {
      initAlloy,
      applyFullPagePropositions
    } = await import("./alloy-utils.js");
    await initAlloy();
    await applyFullPagePropositions();
   }
```

```
// Apply remaining Target propositions BEFORE loading the first section
   // and after initial decoration
   // This prevents flicker by ensuring Target changes are ready to apply
   // as soon as the section is loaded
   if (adobeTarget) {
      const {
        applyRemainingPropositions,
        initSectionDisplayTracking,
      } = await import("./alloy-utils.js");
      await applyRemainingPropositions();
      initSectionDisplayTracking();
    }
```

THE SECOND WAVE

→ Watch a `MutationObserver` on those sections

→ Block-status attributes

→ Apply each remaining activity the *moment its selector resolves* so decisions land as the page builds itself, not before

02 Scope flicker control.

CHALLENGE	The native SDK hides the entire `<body>` until decisions render, which fights EDS's own visibility handling and delays LCP.
SOLUTION	Instead, this code injects per-selector hide rules and removes each one *the instant* its proposition applies.
IMPACT	Targeted flicker mitigation with no performance penalty.

03 Section-aware content handling.

CHALLENGE	Target offers that include a section wrapper would otherwise produce a doubled-up `<div class="section">` on apply, this is a problem as your block code may break due to the extra div section being applied.
SOLUTION	The code detects section-targeted offers and unwraps the outer wrapper *before* handing content to the SDK.
IMPACT	The markup stays clean.

04 Manual impression firing everywhere.

CHALLENGE	The moment you opt out of auto-render, every impression becomes your responsibility.
SOLUTION	The code rebuilds the exact payload the SDK produces internally and fires it after each successful apply—including the durable case where a redirect activity needs the impression to survive a page navigation via `sendBeacon`
IMPACT	Reporting stays accurate without the SDK doing it for us.

FOUNDATION YOU CAN RELY ON

A personalization layer is *only as good* as the authoring workflows it supports.

Solutions that only handle one activity type force marketers into a narrow lane.

That's why the integration contract pattern has been validated across three core activity types that represent real-world personalization—running on both the Adobe Web SDK and Document Authoring.

VISUAL EXPERIENCE COMPOSER (VEC) CHANGES DELIVERED THROUGH TARGET	The click-to-edit experiences marketers build directly applied correctly against EDS's progressively decorated DOM
DOCUMENT AUTHORING FRAGMENTS VIA THE PUSH TO TARGET FEATURE	Content authored in the document-based workflow, pushed into Target, & delivered as section-level offers with clean, single-wrapped markup
PAGE REDIRECT TARGET TESTS	Redirect activities whose impressions fire durably & survive the navigation, so the test reports accurately every time

Turning one-off patches into a reliable architecture.

VEC edits, document-authored fragments, and redirect tests are very different mechanisms under the hood—getting all three working against the same progressive-rendering model is what turns this from a one-off fix into a **dependable foundation.**

WHEN TO USE THIS APPROACH

The honest tradeoffs.

We believe in being straight about engineering decisions, so here's the cost.

Because we own the apply phase, we also own the behaviors the SDK used to guarantee: impression firing, redirect durability, JSON content delivery. Any bug in those manual steps surfaces as an analytics anomaly rather than a hard error, which means monitoring and reporting validation is *critical to the overall solution*—not just an afterthought.

Is there an alternative?

Yes, but the reality is staying fully native is only viable if you forbid block- and section-level personalization entirely—only targeting elements present in the initial HTML. **And, this will strip out most of Target's value on an EDS site.** For our client, and for the kind of sophisticated, on-brand personalization their roadmap called for—it was clear that owning the contract was the right trade off.

Why this matters for *your* roadmap.

Edge Delivery Services is one of the best things to happen to web performance on the Adobe stack in years. But its progressive-rendering model genuinely changes how personalization has to work, and the gap between the documented happy path and a production-grade implementation is wider than most teams expect going in.

That gap is exactly where SHIFT Paradigm does its best work.

We don't just connect Adobe tools, we engineer the contracts between them so that marketers get the full power of tools like Target without sacrificing the performance that drew them to EDS in the first place.

» Standing up personalization on AEM Edge Delivery Services?
Wondering why your Target activities are flickering & missing their marks?

SHIFT PARADIGM

» CONTACT US TO COMPARE NOTES