You’re a JavaScript developer.

Maybe working solo. Maybe leading a team.

Your goal:

Use AI for real impact.

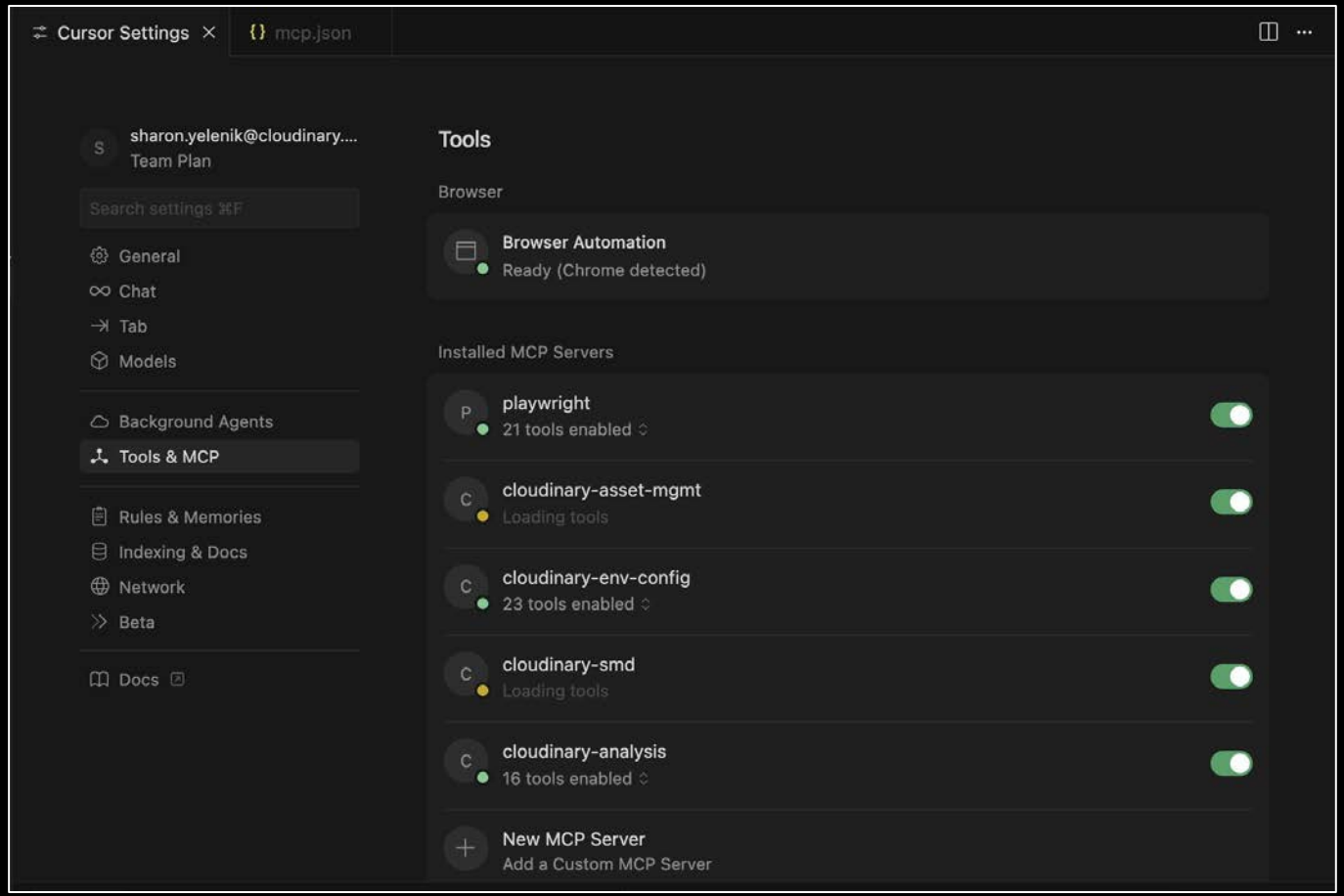
- What does “using AI effectively” actually mean?
- What measurable benefits are we expecting?
- What best practices should we agree on to make sure AI is accurate enough to be trustworthy?

Because there will only be buy-in if it actually works well.

Making API Docs and AI Work Together: Improving Context, Accuracy, and Agentic Coding in the IDE

Sharon Yelenik, Senior DevX & API Content Writer, Cloudinary

MCP Servers within the IDE (Cursor)

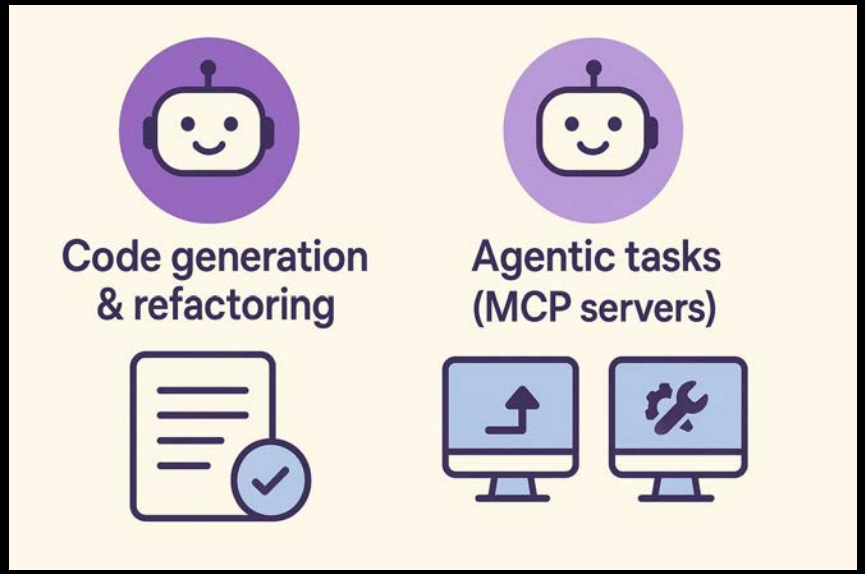


What Does “Using AI Effectively” Actually Mean?

LLM Client – Two Use Cases

The LLM Client is the AI inside the IDE that reads documentation, reasons, and performs actions.

- **Code generation & refactoring** — AI increasing coding efficiency.
- **Agentic tasks** — AI executing actions like uploading assets or configuring environments.
    - Use MCP servers to handle authentication, validation, and execution so AI can act safely.



What Measurable Benefits are We Expecting?

The API Boom

63 % of developers use three or more external APIs, and 28 % use six or more.

- *The State of APIs 2024 by History Tools*

API traffic has skyrocketed — up 800 % between 2016 and 2021 according to Cloudflare.

- *The State of APIs 2024* by History Tools

69 % of developers spend over 10 hours a week *just* working with APIs.

- *The 2025 Postman report*

From **Next.js** to **Auth0**, **Stripe**, **Cloudinary**, and more — APIs are the connective tissue that make modern JavaScript apps work. The LLM client streamlines their use.

Stats on Using AI in Development

Software developers using generative AI tools in their IDEs can complete certain coding tasks **up to twice as fast** compared to the traditional workflow..

- *A McKinsey study*

Developers solved problems *up to 55% faster* when using an AI pair programmer. In practice, this means a dev might finish in hours what previously took a day.

- *AI for Coding: Benefits, Challenges, Tools and Future (Echo)*

68% of developers told Atlassian they're saving at least 10 hours per week using AI tools, including agents.

- *ITPro*

However...

Do Developer Trust in AI?

More developers *distrust* AI tools than trust them:

46% distrust vs 33% trust — and only 3% "highly trust" AI output.

- Stack Overflow Survey 2025

Only a small share of developers (just 3.8%) report experiencing both **low hallucinations** and high confidence in shipping AI-generated code.

- Qodo Report - 2025 State of AI Code Quality

AI Agents Still a Minority

52% of developers don’t use agents yet.
37.9% say they have no plans to.

- *Stack Overflow 2025 Developer Survey*

87% worry about accuracy.
81% worry about data security.

- *Stack Overflow 2025 Developer Survey*

While everyone’s curious, most aren’t ready to delegate real tasks to AI yet.

- *Stack Overflow 2025 Developer Survey*

The Data Is Clear: Missing Context is the Problem

About 65% of developers say AI “misses relevant context” when refactoring.

- *Stack Overflow 2025 Developer Survey*

53% AI “champions” (developers who like using AI) of them still want better context.

- *Stack Overflow 2025 Developer Survey*

When AI hurts code quality, 44% blame missing context.

- *Stack Overflow 2025 Developer Survey*

What's the Opportunity in Improving AI Results?

- **Engineers with high confidence in AI** are 1.3 × more likely to say it makes their job enjoyable.
- **Confidence creates adoption.**

The lesson: The better the context, the better the **LLM client** performs, and the easier it is to get real buy-in from the team.

Improve Context with Docs in the LLM Client

- Documentation and AI are now a team.
- Docs exist to give developers what they need: setup instructions, code snippets, examples.
- AI now needs to use them — in real time — inside the IDE.

In the past, you went to the docs site to look things up.
Now your IDE does that for you.

The LLM depended on the **docs as the source of truth** for how to get things done, the same way a person would.

Image & Video APIs > Get Started > Try it! > Code explorers

Try it! Code explorers

Rate this page:

Open as Markdown | Copy for LLM | Download Markdown

Last updated: Sep-21-2025

Select your preferred language or framework, then choose a code explorer to learn how to implement your own capabilities in minutes.

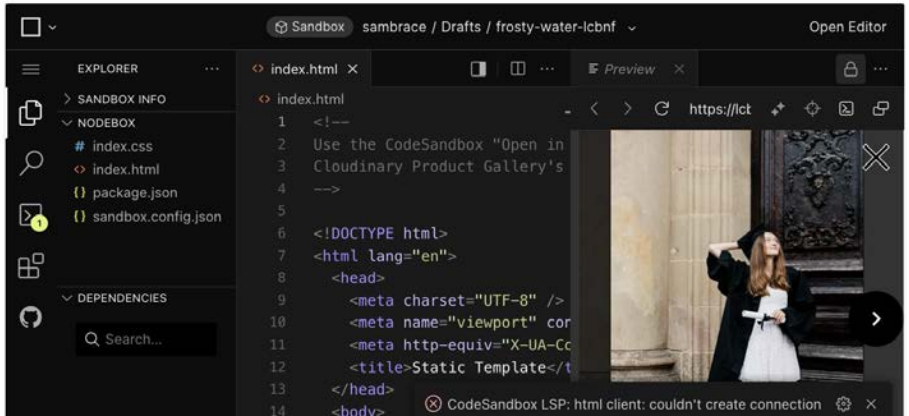
Tip: Can't see your preferred language or framework? Check out all our SDKs and Community-developed libraries.

JavaScript

Create and customize: Product Gallery

Create and customize: Product Gallery

Start with a simple product gallery and uncomment config lines to customize it.



What Best Practices Should We Agree on to Make Sure AI is Accurate Enough to be Trustworthy?

JavaScript APIs

- Apply these principles to: Auth0, Firebase, Supabase, Slack, Shopify, Cloudinary, and many others.
- We'll use Cloudinary for the purposes of demonstration, and revisit the other APIs at the end.

Tip #1: Utilize MCP Servers

How to Add MCP Servers in Cursor:

1. Go to **Cursor > Settings > Tools & Integrations**
2. From the MCP Tools section, click New MCP Server
3. Add the MCP servers to your JSON list, as documented per API:

```
"cloudinary-asset-mgmt-remote": {
  "url": "https://asset-management.mcp.cloudinary.com/sse"
},
"cloudinary-env-config-remote": {
  "url": "https://environment-config.mcp.cloudinary.com/sse"
},
"cloudinary-smd-remote": {
  "url": "https://structured-metadata.mcp.cloudinary.com/sse"
}
```

4. In Cursor Settings, click **Needs Login** for MCP servers that requires authentication.

Tip #2: Reference the API's Doc Site in your IDE

How to Add Docs in Cursor:

Note: If you're using a different IDE, check its MCP or docs integration options. Many of the same principles apply.

1. Go to Cursor > Settings > Indexing & Docs
2. In the Docs section, click + Add Doc
3. Enter your doc site (e.g. `https://cloudinary.com/documentation`)
4. When writing a prompt related to transformations, click the **Add context** (**@**) button and select the pre-defined name of the file you want to use for context.

By anchoring your LLM to the doc site, you ensure its suggestions aren't just valid code, but also reflect the documented best practices for your environment.

Tip #3: Index docs with Context7

- **Context7** provides an **MCP server** that indexes the code snippets and examples from API documentation.This means your **LLM client** can autocomplete and generate code directly from the real documentation, not guesses.

How to Add Context7 in Cursor:

1. Go to **Cursor > Settings > Tools & Integrations**
2. From the MCP Tools section, click New MCP Server
3. Add Context7 to your JSON list of MCP servers:
   `"context7": {"url": "https://mcp.context7.com/mcp"}`
4. Once set up, your LLM client can autocomplete directly from the indexed docs.

To enable Context 7 in a different IDE, see Context 7's GitHub repo README & choose an IDE.

Thousands of products already index their docs in Context7. Make sure yours is included.

Tip #4: Leverage Rules

Why rules?

- Turn **“best practices”** into **executable guidance** the LLM client follows in your IDE.
- Reduce hallucinations by **constraining** suggestions to what’s valid for your stack.
- Standardize outputs across the team (naming, architecture, API usage).

Tip #4: Leverage Rules - Use Provided Rules Files

How to Add a Rules File in Cursor

1. Get the name of the rules file that the product you're working with provides (e.g. `cloudinary_transformation_rules.md`).
2. Go to **Cursor > Settings > Indexing & Docs**
3. In the **Docs** section, click **+ Add Doc**
4. Enter the name of the rules file

When writing a prompt about transformations, add this rules doc as context. In chat, use **@ Add Context → Docs → + Add new doc**, then paste the rules URL.

Tip #4: Leverage Rules - Create Your Own Rules

Where rules live (Cursor)

- Project folder: `/.cursor/rules/*.mdc`
- One project-wide file (optional): `/.cursor/index.mdc`
- Format: Markdown with clear headings + lightweight examples the model can copy.

Tip #4: Leverage Rules

- **What to encode (the "4 C's")**
  - **Compatibility** — note which parameters or methods work together.
  - **Conventions** — your naming, folder, or pattern rules.
  - **Constraints** — what AI must not do (e.g., credentials in client code).
  - **Canon** — small, correct code examples (10–20 lines) for the model to reuse.

- **Guardrails that pay off (Cloudinary examples)**
  - Add `format("auto").quality("auto")` by default
  - Enforce legibility on overlays (white text + dark backdrop)
  - Require `upload_preset` for browser uploads
  - Forbid `g_face` with `c_pad`
  - Limit agentic actions (MCP) to server-side tasks

- **Maintenance Tips**
  - Keep rules short (1–3 pages), review quarterly, and link each to official docs.

Tip #4: Leverage Rules - Cloudinary Rules (.mdc)

See IDE

Tip #4: Leverage Rules - Other IDEs

IDE / Tool	File Extension	Typical Location	Purpose
Cursor	`.mdc`	`/.cursor/rules/`	Markdown rules guiding AI completions
JetBrains AI Assistant	`.md`	`/ai/rules/`	Markdown "Guidelines" for code style & safety
VS Code (AI Extensions)	`.json / .yaml`	`Extension config`	Defines completion constraints
CLI Agents (e.g., Aider)	`.yaml / .toml / .md`	`Repo root`	Rules & prompt templates

Cloudinary Example: Upload Using MCP Servers

- Agentically upload the image you want to use in your website from a remote URL.
- Save context switching by staying within your IDE.

Cloudinary Example: Hero Image

- Use the image you uploaded as the hero image for your website.
- Crop this, and all hero images, consistently at 1600×900.

Cloudinary Example: Image Delivery (URL Gen)

Cloudinary example: Frontend image delivery (URL generation)

- Add a text overlay to the `fathers_day_banner` image, using company standards.

Cloudinary Example: Transform Using Rules Files

- Create campaign product thumbnails for your Father's Day campaign.
- Auto-cropped images around the main subject, following best practices.

Modern APIs Are Becoming Agentic

API	MCP Support	Context7 Indexed	Agentic Examples
Auth0	✅	✅	Set up login and authentication flows, manage test users, or refresh API keys securely.
Firebase	✅	✅	Create or seed a test database, set authentication rules, or deploy to the Emulator Suite from your IDE.
Supabase	✅	✅	Create database tables, run migrations, or publish backend functions without leaving the editor.
Slack	✅	✅	Send team notifications, create channels, or post build and test updates automatically.
Shopify	✅	✅	Add sample products, draft test orders, or connect webhooks for your store.
Cloudinary	✅	✅	Upload media, apply transformations, and manage presets or folders directly from your IDE.

Core Development Tasks for Rules Files

- **Component Scaffolding Rules** – how to create a new React/Vue/Svelte component with default props, structure, and test files.
- **API Fetching Rules** – how to call APIs using fetch or Axios with consistent error handling and no hard-coded keys.
- **Form Handling Rules** – preferred validation library, structure for controlled inputs, and submit patterns.
- **Routing Rules** – how to register new routes in Next.js or Express (naming, import order, layout).
- **State Management Rules** – whether to use Redux, Zustand, or Context API and how to structure stores.

Now You Can Leverage Your LLM Client **Accurately** and **Effectively**!

- Using it for code generation and agentic tasks
- Being confident that it's correct when implementing doc-friendly techniques

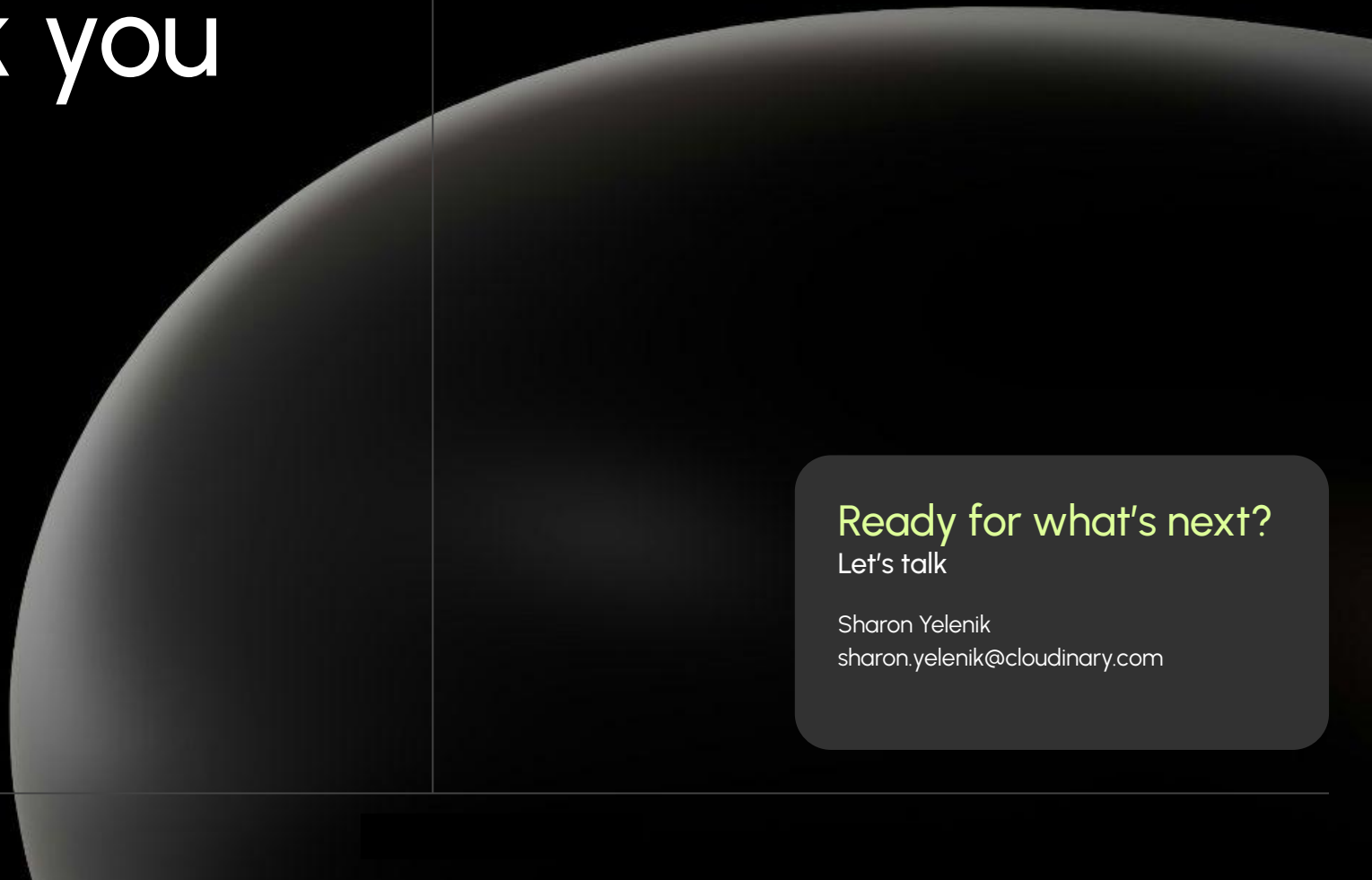
What's Next?

Docs are about to stop being *reference pages* and start being *runtime components.* They'll be adaptive, dynamic, and self-improving.

So the next generation of docs will:

- Live inside our IDEs.
- Power reasoning and execution.
- Evolve dynamically from real usage.

Thank you



Ready for what's next?

Let's talk

Sharon Yelenik
sharon.yelenik@cloudinary.com

Tip #4: Leverage Rules - Use Provided Rules Files

- **More likely to happen without rules in your IDE:**

```
// Invalid: g_face doesn't work with c_pad

https://res.cloudinary.com/demo/image/upload/
c_pad,g_face,w_500,h_500/e_upscale/f_auto/q_auto/
fathers_day_banner.jpg
```

- **More likely to happen *with* rules):**

```
// Valid: g_face with c_auto

https://res.cloudinary.com/demo/image/upload/
c_auto,g_face,w_500,h_500/e_upscale/f_auto/q_auto/
fathers_day_banner.jpg
```

Tip #2: Reference the API's Doc Site in your IDE

- **More likely to happen without docs in your IDE (add transformation directly to URL):**

```
https://res.cloudinary.com/demo/image/upload/w_1600,h_900,c_fill,g_auto/f_auto/q_auto/fathers_day_banner.jpg
```

- **More likely to happen *with* docs (one-time named transformation setup):**

```
import { v2 as cloudinary } from "cloudinary";

cloudinary.api.create_transformation("campaign_hero",
   {width: 1600, height: 900, crop: "fill",
    gravity: "auto",
   fetch_format: "auto",  quality: "auto"})
```

Then, when coding your delivery URLs, just reference the named transformation (best practice):

```
https://res.cloudinary.com/demo/image/upload/t_campaign_hero/fathers_day_banner.jpg
```

Tip #3: Index docs with Context7

- **More likely to happen without Context7 (looks plausible, but wrong in the browser):**

```
// Invalid in the browser — wrong method name, server SDK syntax
const cld = new Cloudinary({ cloudName: "demo" });
const img = cld.image("fathers-day-banner");
img.overlayText("Happy Father's Day!"); // nonexistent API
document.getElementById("banner").src = img.url;
```

- **More likely to happen *with* Context7:**

```
import { Cloudinary } from "@cloudinary/url-gen";
import { overlay } from "@cloudinary/url-gen/actions/overlay";
import { text } from "@cloudinary/url-gen/qualifiers/source";
import { TextStyle } from "@cloudinary/url-gen/qualifiers/textStyle";
import { south } from "@cloudinary/url-gen/qualifiers/gravity";

const cld = new Cloudinary({ cloud: { cloudName: "demo" } });

const banner = cld.image("fathers-day-banner")
  .overlay(
    overlay(
      text("Happy Father's Day!", new TextStyle("Arial", 200))
        .textColor("white").backgroundColor("rgba:00000050")
    ).gravity(south()) // place near the bottom
)
document.getElementById("banner").src = banner.toURL();
```