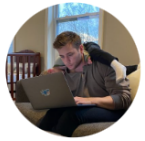
Building your app on Cloudflare

Pick 3: Good, Fast, Cheap

Jordan Finneran
Pimsical

@JordanFinners



brandon
@burcs

the cloudflare dilemma, you can only pick 3:

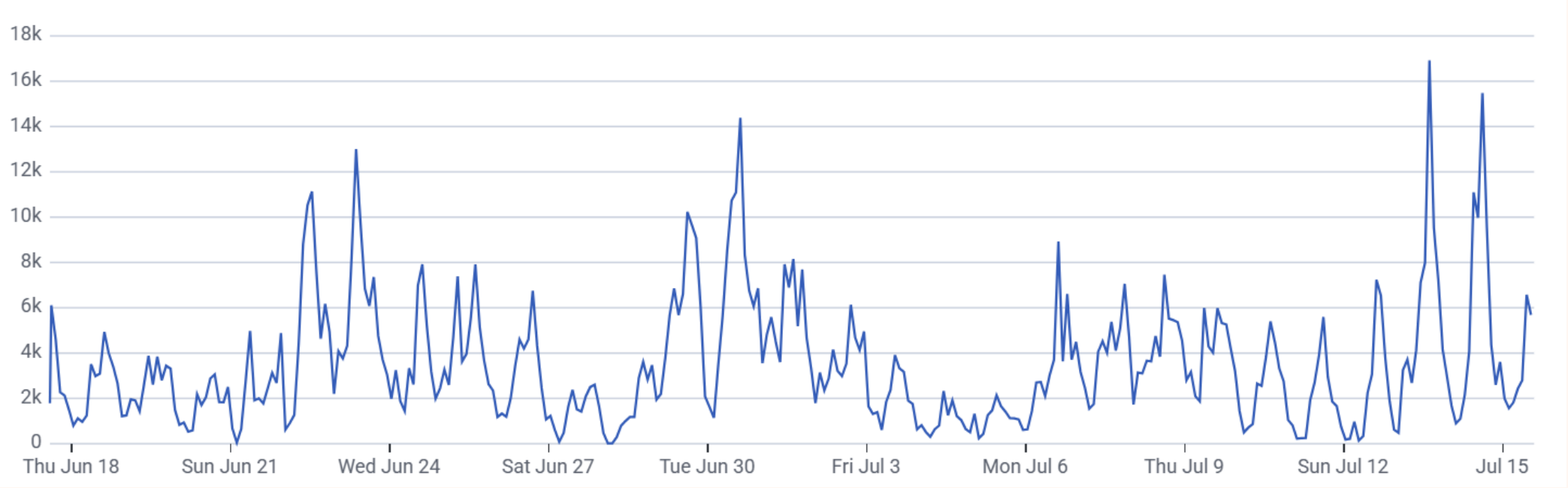
good
fast
cheap

6:57 PM · June 28, 2026

Building your app on Cloudflare

As a solo engineer and having worked in small teams, I need…

- Scale
- Cost
- Dev Speed
- Performance
- Reliability



Who am I & what is Pimsical?

Working in Shopify space for 6+ years

- Stock Take (Inventory Count) app on Shopify POS
- Staff Discounts app
- Retail & Warehouse integration apps
- Built for Shopify
- Build Award Winner - Community 2025

Who cares about their app being:

Good?

Fast?

Cheap?

Who likes their hosting bill?

Could it be free...

or $5 per month

How?

Cloudflare Developer Platform

- **Generous free tier**
- **Paid starting at $5**
- **Excess compute capacity**

Workers Free

Experience the Workers ecosystem without commitment.

$0/mo

100,000 requests per day (Across all of your Worker scripts, UTC+0)

Get started

WORKERS FEATURES

- Includes 100k requests per day
- 10ms CPU time per request
- Access to Developer Platform products. Limits apply

Workers Paid

Build scalable, production ready applications without usage limits.

Starting at $5/mo

$0.30/million requests per month

Get started

WORKERS FEATURES

- 10 million requests/month included + $0.30 per additional million
- 30 million CPU milliseconds/month included, $0.02 per additional million CPU milliseconds
- Access to Developer Platform products

Cloudflare Workers Platform

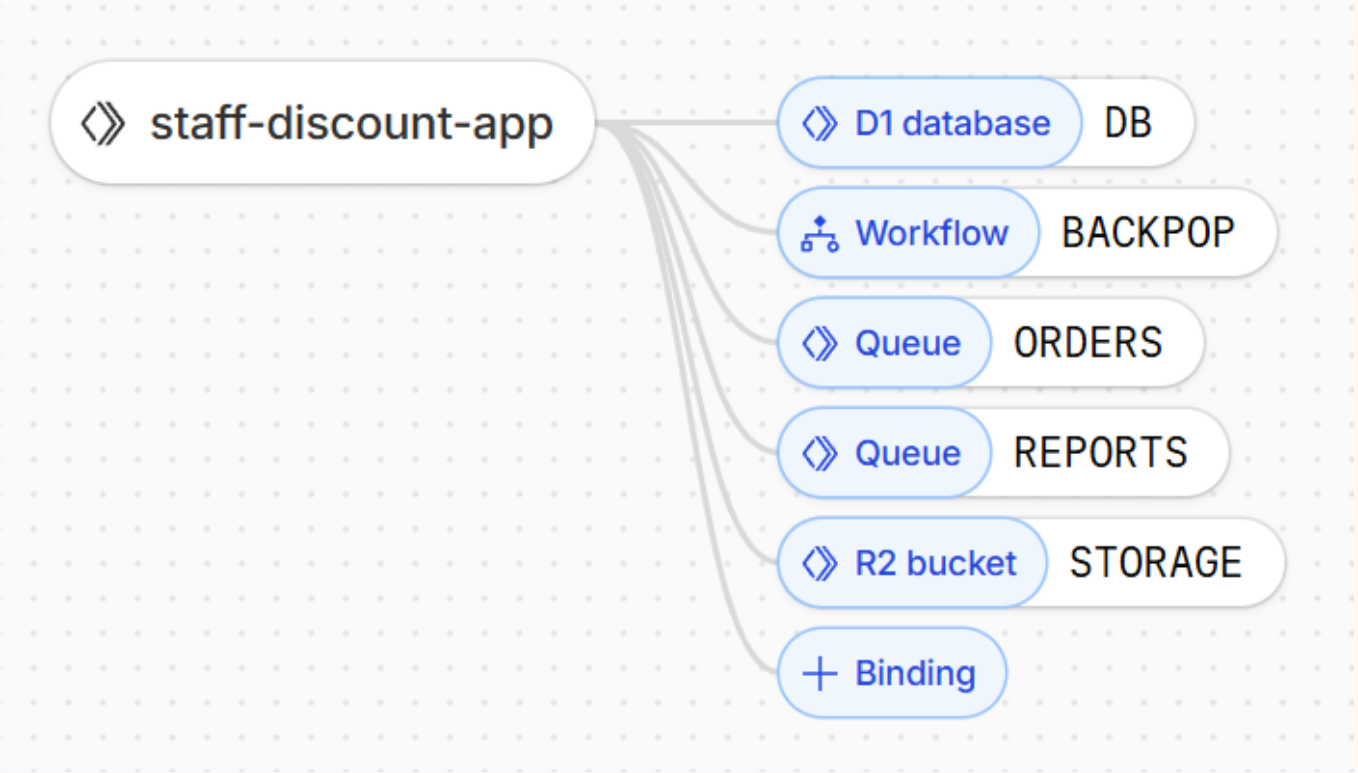
Has rapidly expanded, now covering:

- Workers – Website hosting & Serverless Functions
- R2 Blob Storage – S3 API compatible, no egress bandwidth fees
- D1 Database – SQLite database, truly serverless
- Queues – Async processing, guaranteed delivery,
- Workflows – Durable multi step execution engine
- Durable Objects – Coordinate multiple clients, with private, transactional & strongly consistent storage
- KV – Low-latency, key-value data storage
- AI – Run machine learning models, powered by serverless GPUs

Workers

Speedy, serverless compute
And static hosting

- Building blocks when designing apps
- Built on top of V8 Engine, isolates & browser standard APIs
- Zero cold start
- Distributed to Cloudflare’s global network
- Automatically works with frameworks



Workers

Think AWS Lambda like

- **Processes:**
  - **HTTP Requests**
  - **Queue Messages**
  - **Scheduled Triggers**

- **Supporting:**
  - **Static Assets**
  - **Client Side Rendered**
  - **Server Side Rendered (SSR / ISR etc)**

```
import welcomeHTML from "welcome.html";

export default {
    /**
     * Handle HTTP Requests
     * @param {Request} request the HTTP Request
     * @param {Env} env the Cloudflare Env
     * @returns {Promise<Response>}
     */
    async fetch(request, env) {
        const url = new URL(request.url);
        if (url.pathname === "/api") {
            // You could also call a third party API here
            const data = await import("./data.js");
            return new Response(JSON.stringify(data), {
                headers: {
                    "content-type": "application/json",
                },
            });
        }

        return new Response(welcomeHTML, {
            headers: {
                "content-type": "text/html",
            },
        });
    }
};
```

Workers Limits

Not so fast…

- Free Plan - max 10ms of active CPU time
- Paid Plan – higher active CPU time limits
- Limits on the NodeJS API & Packages you can use
- Language support JS, Python, Rust, WASM
- Observability
- Configured via TOML or JSON file
- Smart placement
- Deploy with 1 command

```toml
name = "pimsical-myapp"
compatibility_date = "2026-07-01"

[[routes]]
pattern = "myapp.pimsical.app"
custom_domain = true

[assets]
directory = "web/dist"
not_found_handling = "single-page-application"
```

```toml
name = "pimsical-api"
compatibility_date = "2026-07-01"
main = "index.ts"

[[routes]]
pattern = "api.pimsical.app"
custom_domain = true

[observability]
enabled = true

[[d1_databases]]
binding = "DB"
database_name = "MY_APP_DB"
database_id = "X"
```

D1:

SQLite Serverless Database

D1

Actually, serverless database…

- Managed, serverless database
- Without any cold start or spin up time
- SQLite semantics
- Built in disaster recovery
- Designed for horizontal scale databases
- Max size of 10GB
- Don’t pay for hours or compute when you aren’t querying your database
- Access from Workers or via HTTP API

```
npx wrangler d1 create MY_DATABASE

npx wrangler d1 migrations apply MY_DATABASE --remote

npx wrangler d1 execute MY_DATABASE --remote --json
--command='SELECT * FROM MY_TABLE'
```

D1 Examples

Example migration file

```sql
-- Migration number: 0001    2024-05-26T19:26:45.357Z
CREATE TABLE STOCK_TAKES (
    id TEXT NOT NULL,
    storeID TEXT NOT NULL,
    name TEXT NOT NULL,
    createdAt DATE,
    createdBy TEXT NOT NULL,
    PRIMARY KEY (id, storeID)
);

```

D1 Examples

Example inserting data from Worker

```
1
2     const stmt = env.DB.prepare(`
3     INSERT INTO STOCK_TAKES(id, storeID, name, createdAt, createdBy)
4     VALUES(?1, ?2, ?3, ?4, ?5)`);
5
6     const { error } = await stmt.bind(
7         crypto.randomUUID(),
8         storeID,
9         stockTakeName,
10        new Date().toISOString(),
11        userID
12    ).run()
13
14    if (error) {
15        return Promise.reject(error);
16    }
17
18    return;
```

D1 Examples

Example querying data from Worker

```
const stmt = env.DB.prepare(`
SELECT * FROM STOCK_TAKES
WHERE storeID = ?1 AND id = ?2`);

const { results, error } = await stmt.bind(
  storeID,
  id,
).all();

if (error) {
    return Promise.reject(error);
}

return results
```

Workflows:

Background Jobs

Workflows

Background jobs…

- Build applications and logic that can run over minutes, hours, days or weeks
- Durable multi step execution engine
- Automatic retries
- Persisted state
- Defined in JavaScript

```
const instance = await env.MY_WORKFLOW.create({
  id: newId,
  params: {some: 'data'}
});

const status = await instance.status()

const myInstance = await env.MY_WORKFLOW.get(instance.id)

```

Workflow Example

```ts
export class OnboardingWorkflow extends WorkflowEntrypoint<Env> {
    async run(event: WorkflowEvent, step: WorkflowStep) {
        // Can access bindings on `this.env`
        // Can access params on `event.params`
        const { email, storeID } = event.payload
            await step.sleep('onboarding wait', '2 minutes')

            await step.do('welcome email', async () => {
                return sendTemplateEmail(this.env, email, EmailTemplate.WELCOME, { StoreURL: storeID })
            })

            await step.sleep('get started wait', '1 day')

            const hasSentUninstalled = await step.do('get started email', {
                retries: {
                    limit: 1,
                    delay: "30 seconds",
                    backoff: "exponential"
                },
                timeout: "1 minute",
            }, async () => {
                const [isInstalled, hasScannedItems] = await Promise.all([
                    isStoreValid(this.env, storeID),
                    hasScanned(this.env, storeID),
                ])
                if (!isInstalled) {
                    await sendTemplateEmail(this.env, email, EmailTemplate.UNINSTALLED, { StoreURL: storeID
})
                    return true
                }

                if (hasScannedItems) {
                    return false
                }


                await sendTemplateEmail(this.env, email, EmailTemplate.GET_STARTED, { StoreURL: storeID })
                return false
            })

            if (hasSentUninstalled) {
                return
            }

            await step.sleep('uninstalled wait', '5 days')
            // some more logic
```

Workflow Example

```ts
import {
  WorkflowEntrypoint, WorkflowStep, WorkflowEvent
} from 'cloudflare:workers';

export class OnboardingWorkflow extends WorkflowEntrypoint<Env> {
    async run(event: WorkflowEvent, step: WorkflowStep) {
        // Can access bindings on `this.env`
        // Can access params on `event.params`
    }
}
```

Workflow Example

```
async run(event: WorkflowEvent, step: WorkflowStep) {
    // Can access bindings on `this.env`
    // Can access params on `event.params`

    const { email, storeID } = event.payload
    await step.sleep('onboarding wait', '2 minutes')

    await step.do('welcome email', async () => {
        return sendTemplateEmail(this.env, email)
    })
}
```

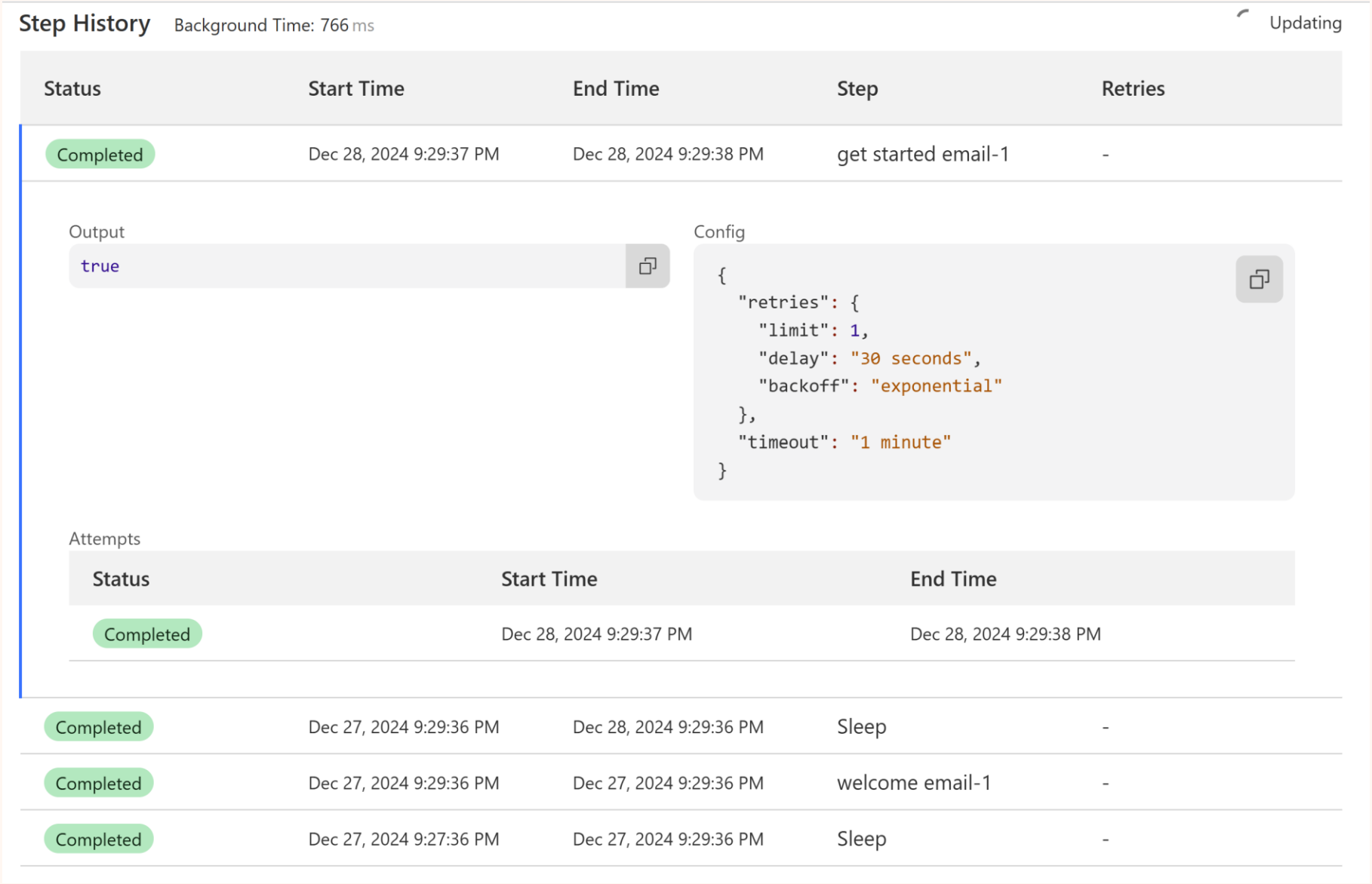
```
async run(event: WorkflowEvent, step: WorkflowStep) {
    // previous logic
    await step.sleep('get started wait', '1 day')

    const hasSent = await step.do('get started email', {
            retries: {
                limit: 1,
                delay: "30 seconds",
                backoff: "exponential"
            },
            timeout: "1 minute",
        }, async () => {
            const hasScannedItems = await hasScanned(
              this.env,
              storeID
            )
            if (hasScannedItems) {
                return false
            }

            await sendTemplateEmail(this.env, email)
            return true
        })

    console.log(hasSent)
    await step.sleep('uninstalled wait', '5 days')
    // some more steps
}
```

Workflow Example

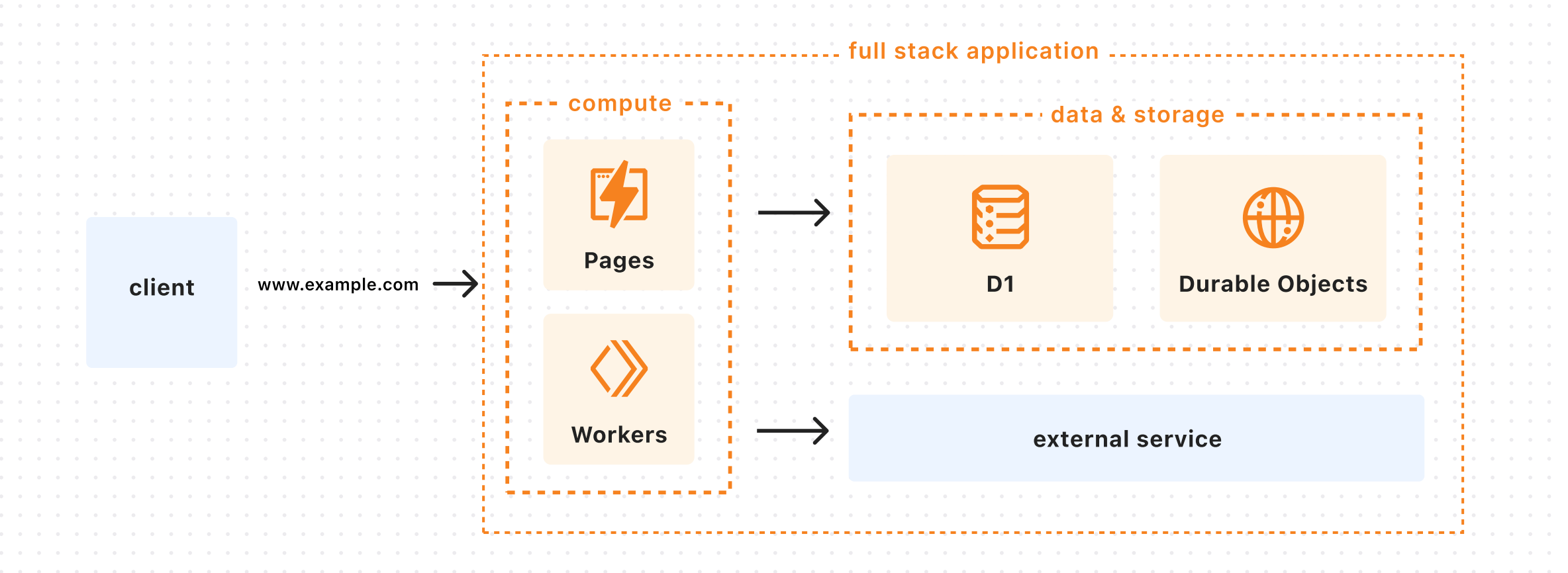


Example:

What does an application look like?

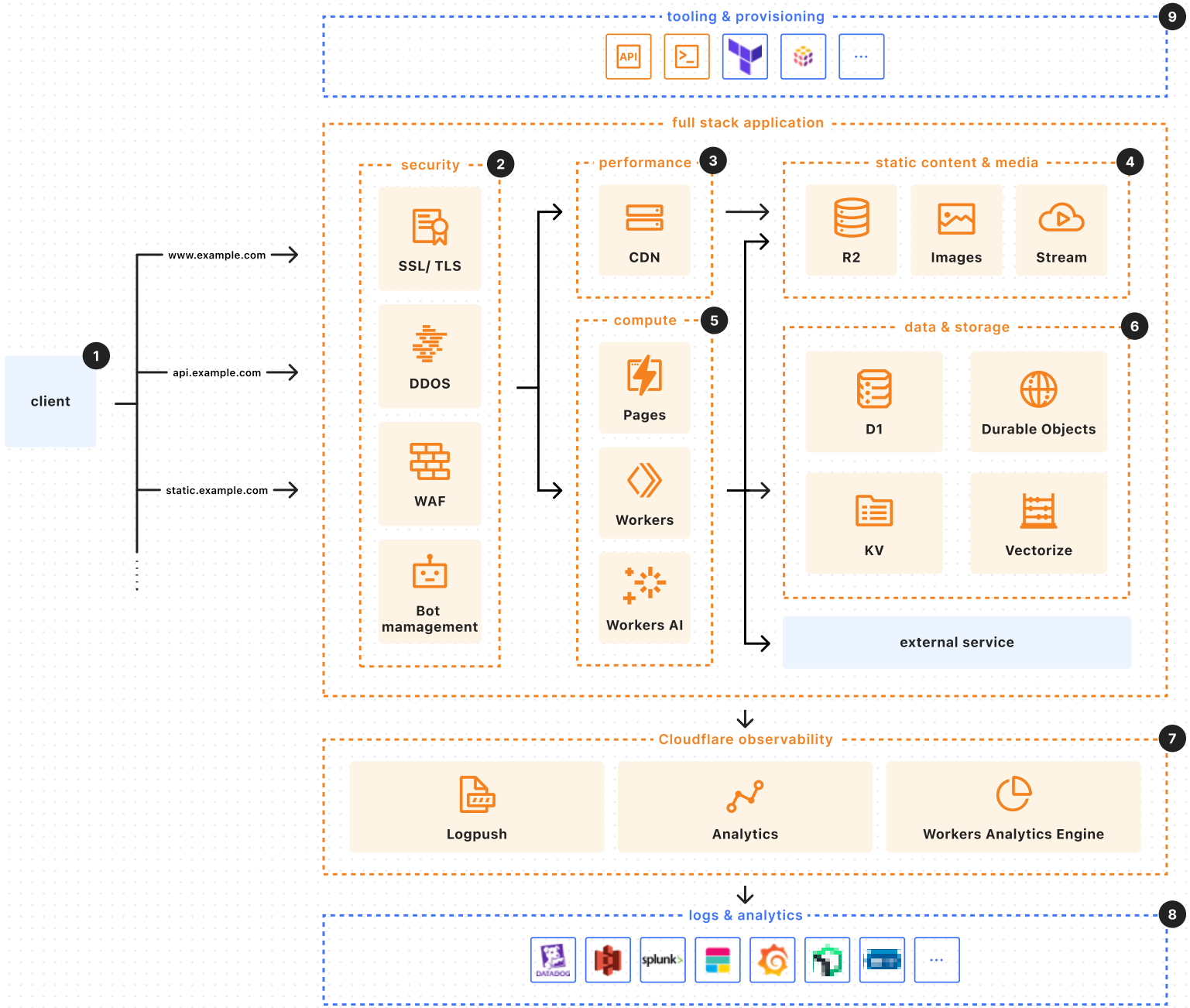
Example

Example application...



Example

Example full stack application...



FULLSTACK APPLICATION

CLOUDFLARE

Who is trying Cloudflare?

Thank you

Jordan Finneran
Pimsical

@JordanFinners