Let's do this in **4 stages** using a tiny Node.js CRUD API and then wire it into **Cypress Cloud Smart Orchestration**:

1. Build a simple Node.js CRUD app
2. Add Cypress tests & split them into multiple specs
3. Connect to Cypress Cloud and enable **parallelization + load balancing**
4. Turn on **spec prioritization** and **auto cancellation** in CI

I'll assume you're okay with npm and basic terminal commands.

1️⃣ Simple Node.js CRUD API (items)

Step 1: Create the project

```
mkdir smart-orchestration-demo
cd smart-orchestration-demo
npm init -y
npm install express
```

Step 2: Create `server.js`

```
// server.js
const express = require('express');
const app = express();
const PORT = 4000;

app.use(express.json());

// In-memory "database"
let items = [];
let currentId = 1;

// CREATE
app.post('/api/items', (req, res) => {
  const { name } = req.body;
  if (!name) return res.status(400).json({ error: 'Name is required' });
```

```
  const newItem = { id: currentId++, name };
  items.push(newItem);
  res.status(201).json(newItem);
});

// READ ALL
app.get('/api/items', (req, res) => {
  res.json(items);
});

// READ ONE
app.get('/api/items/:id', (req, res) => {
  const id = Number(req.params.id);
  const item = items.find(i => i.id === id);
  if (!item) return res.status(404).json({ error: 'Item not found' });
  res.json(item);
});

// UPDATE
app.put('/api/items/:id', (req, res) => {
  const id = Number(req.params.id);
  const { name } = req.body;

  const item = items.find(i => i.id === id);
  if (!item) return res.status(404).json({ error: 'Item not found' });
  if (!name) return res.status(400).json({ error: 'Name is required' });

  item.name = name;
  res.json(item);
});

// DELETE
app.delete('/api/items/:id', (req, res) => {
  const id = Number(req.params.id);
  const index = items.findIndex(i => i.id === id);
  if (index === -1) return res.status(404).json({ error: 'Item not found' });

  items.splice(index, 1);
  res.status(204).send();
});

app.listen(PORT, () => {
  console.log(`API running on http://localhost:${PORT}`);
});
```

Step 3: Run the API

Add a script in `package.json`:

```
"scripts": {
  "start": "node server.js"
}
```

Run it:

```
npm start
# API at http://localhost:4000
```

2️⃣ Add Cypress tests (split into multiple specs)

Install Cypress:

```
npm install cypress --save-dev
npx cypress open
```

This creates `cypress/` folder and `cypress.config.js` (if not, Cypress will guide you).

Step 1: Basic Cypress config

In `cypress.config.js`:

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:4000',
    setupNodeEvents(on, config) {
      // you can add tasks/plugins here later
      return config;
    },
  },
});
```

Step 2: Create multiple specs (important for parallelization!)

Create **three spec files**:

cypress/e2e/create_read.cy.js

```
describe('Create & Read Items', () => {
  it('creates an item', () => {
    cy.request('POST', '/api/items', { name: 'Item A' })
      .its('status')
      .should('eq', 201);
  });

  it('lists items', () => {
    cy.request('GET', '/api/items')
      .its('status')
      .should('eq', 200);
  });
});
```

cypress/e2e/update.cy.js

```
describe('Update Items', () => {
  it('updates an existing item', () => {
    cy.request('POST', '/api/items', { name: 'Item B' }).then((res) => {
      const id = res.body.id;

      cy.request('PUT', `/api/items/${id}`, { name: 'Item B Updated' })
        .its('status')
        .should('eq', 200);
    });
  });
});
```

cypress/e2e/delete.cy.js

```
describe('Delete Items', () => {
  it('deletes an existing item', () => {
    cy.request('POST', '/api/items', { name: 'Item C' }).then((res) => {
      const id = res.body.id;

      cy.request('DELETE', `/api/items/${id}`)
        .its('status')
        .should('eq', 204);
    });
```

```
  });
});
```

Step 3: Run tests locally (serial)

Add script:

```
"scripts": {
  "start": "node server.js",
  "cy:run": "cypress run"
}
```

Run:

```
npm start       # terminal 1
npm run cy:run  # terminal 2
```

So far: tests run **one after another** on your machine.

3️⃣ Cypress Cloud + Smart Orchestration (Parallel + Load Balancing)

Now we move to **Cypress Cloud**. Smart Orchestration uses **parallelization, load balancing, spec prioritization, and auto cancellation** to optimize CI resources. (docs.cypress.io)

Step 1: Create Cypress Cloud project

1. Go to Cypress Cloud site and log in. (cypress.io)
2. Create a new project.
3. It will give you:
    - a **projectId**
    - a **record key** (Cypress record key).

Step 2: Add `projectId` to `cypress.config.js`

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  projectId: 'YOUR_PROJECT_ID_HERE',
  e2e: {
    baseUrl: 'http://localhost:4000',
  },
});
```

Step 3: Store the record key securely

In CI, you'll put this in an env var like `CYPRESS_RECORD_KEY`. Locally you can test with:

```
export CYPRESS_RECORD_KEY=your-real-key
```

Step 4: Run tests with recording + parallelization

Cypress Cloud parallelization is enabled via CLI flags:

```
npx cypress run --record --key $CYPRESS_RECORD_KEY --parallel
```

This command:

- **Uploads** run data to Cypress Cloud
- **Enables Smart Orchestration**:
    - **Parallelization**: tests can be run on multiple machines
    - **Load balancing**: Cypress Cloud decides which spec file runs where, dynamically, based on history (docs.cypress.io)

> On a **single** machine, `--parallel` doesn't speed things up much. The real power comes when you run the same command on **multiple CI machines** for the same commit.

4️⃣ Example CI setup (GitHub Actions) for parallelization

Let’s say your code is on GitHub.

Step 1: Add workflow file

Create `.github/workflows/cypress.yml`:

```
name: Cypress Tests

on:
  push:
    branches: [ main ]
  pull_request:

jobs:
  cypress-tests:
    runs-on: ubuntu-latest

    strategy:
      fail-fast: false
      matrix:
        containers: [1, 2]   # 2 parallel machines

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Use Node
        uses: actions/setup-node@v4
        with:
          node-version: 20

      - name: Install dependencies
        run: npm install

      - name: Start API
        run: npm start &

      - name: Wait for API
        run: npx wait-on http://localhost:4000/api/items || sleep 5

      - name: Cypress run (parallel)
        env:
          CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
        run: |
```

```
npx cypress run \
  --record \
  --parallel \
  --key $CYPRESS_RECORD_KEY \
  --ci-build-id ${{ github.run_id }} \
  --group "GitHub Actions - Container ${{ matrix.containers }}"
```

What's happening:

- `matrix.containers: [1, 2]` → runs **two jobs in parallel**.
- Each job runs `npx cypress run --record --parallel ...`
- Cypress Cloud:
    - Knows these jobs belong to **one build** (via `--ci-build-id`)
    - **Distributes spec files** dynamically using historical run times (load balancing). (docs.cypress.io)

So:

- First run: specs might be split more or less equally.
- Next runs: Cloud learns which specs are slow and reorders them so machines finish at almost the same time → less idle time.

5️⃣ Spec Prioritization (run important specs first)

Spec Prioritization is part of Smart Orchestration, mainly available for **Business/Enterprise plans**. It tells Cypress Cloud to run **recently failing specs first**, so you get faster feedback on the risky parts. (docs.cypress.io)

Step 1: Enable in Cypress Cloud

1. Open your project in Cypress Cloud.
2. Go to **Project Settings**.

3. Find **Smart Orchestration** section.
4. Toggle **Spec Prioritization** ON. (docs.cypress.io)

Now when your CI runs with `--record --parallel`, Cypress Cloud:

- Looks at **previous run history**
- Runs **last failed specs first**, then others

So if `update.cy.js` was failing in the last run, it will be pushed to the **front of the queue** on the machines.

> For you as a beginner: no change in code/commands. It's a **Cloud setting** only.

6️⃣ Auto Cancellation (stop redundant pipelines)

When you push many commits quickly, older CI runs become useless. **Auto Cancellation** cancels:

- Old test runs / pipelines for the same branch
- Once a newer commit's run has started

This saves CI resources and speeds feedback. (cypress.io)

Typical setup patterns

Depending on your CI:

- **GitHub Actions**:
    - Use `concurrency` + Cloud Auto Cancellation together.

Example:

```
concurrency:
  group: cypress-${{ github.ref }}
  cancel-in-progress: true
```

This makes GitHub auto-cancel older workflows for the same branch. Cypress Cloud then marks those runs as cancelled.

- In **Cypress Cloud**:
    - Some tiers offer extra Auto Cancellation controls (e.g., cancel older runs when new run for same branch starts, or cancel after fail-fast). (testomat.io)

Again, **your Cypress command does not change**. You still run:

npx cypress run --record --parallel --key $CYPRESS_RECORD_KEY

Auto cancellation is handled by CI + Cloud settings.

7️⃣ Quick mental model for you

For your simple Node.js CRUD app:

- You write multiple specs:
    - `create_read.cy.js`
    - `update.cy.js`
    - `delete.cy.js`
- In CI, you run:
    - Same command on **2+ machines** with `--record --parallel`
- Cypress Cloud does:

1. **Parallelization**
    - Multiple machines run specs at the same time → total time drops. (frugaltesting.com)

2. **Load Balancing**
    - It uses historical spec duration to decide which spec runs where, to avoid idle machines. (docs.cypress.io)
3. **Spec Prioritization**
    - Recently failing spec files run first → you see important failures earlier. (docs.cypress.io)
4. **Auto Cancellation**
    - Older runs are cancelled when a newer run starts, so CI doesn't waste time on outdated commits. (cypress.io)