【Next.js】 部署模式總覽

下面整理 **Next.js 的主要部署類型**、每種模式的特性與常見應用場景，並特別解釋 **standalone** 是什麼以及它適合的情境。

---

1 Next.js 部署模式總覽

部署類型	`next.config.js` / 指令	產物	特性與適用場景
1. Node Server (傳統模式)	`next build && next start`	`.next/`	**預設**，完整支援 SSR、ISR、API Routes。部署到任何 Node 環境（PM2、Docker、K8s、AWS EC2、GCP、Heroku...）。
2. Standalone	`output: 'standalone'` + `next build`	`.next/standalone` + `.next/static`	只輸出**最小可執行檔**（server.js + 必要檔案），方便 Docker/容器化。功能與 Node Server 相同（SSR/ISR/API Routes 皆可）。
3. Static Export	`next export` 或 `output: 'export'`	`out/`	完全靜態 HTML/CSS/JS。無 SSR/ISR/API Routes。適合純內容網站、部落格、文件站。
4. Edge Runtime / Vercel	`runtime: 'edge'` 或自動判斷	Vercel Edge Functions (或 Cloudflare Workers)	以 Web 標準 API 為基礎，無 Node built-ins。啟動極快、全球節點分布，適合 Middleware、即時驗證、A/B 測試。
5. Serverless Functions	Vercel / AWS Lambda	Lambda zip	每個頁面/Route 轉成獨立 Serverless Function。可自動擴展，適合不維護長駐伺服器的架構。
6. Custom Server	自行建立 `server.js` 並 `next()`	`.next/`	自定義 Node 伺服器，整合 Koa/Express/Fastify，做複雜路由或中介層。

> 註：模式之間可以組合，例如 **Node Server + Standalone**、**Vercel Serverless + Edge Middleware**。

模式	需 Node.js Runtime	常見部署基底	代表場景
Node Server	✅	Node image / VM	全功能 SSR/ISR + API
Standalone	✅ (但包體精簡)	Node image / VM	Docker/K8s 輕量部署
Static Export	❌	Nginx、GitHub Pages、S3+CDN	純靜態內容
Edge Runtime	❌ (Web Worker API)	Vercel Edge / Cloudflare Workers	全球低延遲運算、Middleware
Serverless Functions	✅ (平台提供)	AWS Lambda / Vercel Functions	無伺服器、自動擴展
Custom Server	✅	Node image / VM	自訂路由、混合架構

2 各模式詳解

① Node Server (預設)

- **啟動**：`next build && next start`
- **產物**：`.next/` 目錄
- **特性**：完整支援 SSR、ISR、API Routes。
- **適用**：自行管理伺服器（EC2、Kubernetes、Docker Swarm、Heroku...）。
- **優缺**：最彈性，但部署包體積較大，需要完整 node_modules。

---

② Standalone

- **設定**：

```
// next.config.js
module.exports = {
  output: 'standalone',
}
```

- **產物**：

```
.next/standalone      # 精簡 Node 專案，可直接 node server.js
.next/static          # 靜態資源
```

- **原理**：
  - `next build` 會分析依賴，只把執行需要的檔案、node_modules 打包進 `standalone`。
  - 不再需要完整專案原始碼，也不必把整個 node_modules 打進 Docker。
- **適用場景**：
  - Docker / 容器化：
    只複製 `.next/standalone` + `.next/static`，image 更小、build 更快。
  - Kubernetes / ECS：
    配合多階段 Docker build，把 `RUN npm ci` 階段與最終執行環境分離。
- **功能**：與 Node Server 相同（SSR / ISR / API Routes 全支援）。

---

③ Static Export

- **指令**：`next build && next export`
- **產物**：`out/` 完全靜態
- **限制**：無 SSR、無 API Routes、無 ISR。
- **適用**：
  - 部落格、說明文件、Landing Page
  - 部署到任何靜態空間（S3+CloudFront、GitHub Pages、Netlify）。

---

④ Edge Runtime / Middleware

- **用法**：在頁面或 route handler 中 `export const runtime = 'edge'`
- **部署**：Vercel Edge Functions、Cloudflare Workers、Deno Deploy。
- **特性**：全球節點極低延遲；不能用 Node built-ins。
- **適用**：
  - 驗證/權限檢查
  - A/B 測試、URL 轉向
  - 即時 Header/Cookie 操作

---

⑤ Serverless Functions (Vercel / AWS Lambda)

- **部署**：Vercel 自動化，或 `serverless-http` 將 Next 整包轉為 Lambda。
- **特性**：
  - 每個頁面/路由轉成獨立 Lambda。
  - 自動擴展、免維運。
- **適用**：
  - 高併發但流量不穩定。
  - 只需 Node 短期計算、不需長連線。

---

⑥ Custom Server

- **方式**：建立 `server.js`

```
const express = require('express')
const next = require('next')
const app = next({ dev: false })
const handle = app.getRequestHandler()
app.prepare().then(() => {
  const server = express()
  server.get('/health', (req, res) => res.send('ok'))
  server.all('*', (req, res) => handle(req, res))
  server.listen(3000)
})
```

- **適用**：
  - 需要與既有 Node 架構整合。
  - 自訂路由、中介層或 WebSocket。

---

3 如何選擇

需求	建議模式
企業內部、自管 Docker/K8s	**Standalone (Node Server)**：產物小、部署快
純靜態內容、無後端邏輯	**Static Export**
全球極低延遲、近使用者運算	**Edge Runtime / Middleware**
彈性擴展、免維運	**Serverless Functions (Vercel / AWS Lambda)**
需與自訂 Node 架構整合	**Custom Server**

總結

- **Standalone** 是 **Node Server 模式的輕量化打包**，非常適合容器化部署。
- 如果要保留完整 SSR、ISR、API Routes，又希望 **映像檔小、啟動快**，選 **Standalone** 幾乎是最佳解。
- 其他模式（Static Export、Edge、Serverless）則依你的應用是否需要即時運算、全球分佈或純靜態內容來決定。

Next.js 部署模式功能支援度

部署模式	需要 Node Runtime	SSR	ISR	API Routes	適用場景
Node Server（`next build && next start`）	✅	✅	✅	✅	傳統部署，功能完整。適合自管伺服器、Docker、K8s。
Standalone（`output: 'standalone'`）	✅	✅	✅	✅	與 Node Server 相同，但打包精簡，適合容器化。
Static Export（`next export`）	❌	❌	❌	❌	只能輸出純 HTML/JS/CSS。適合靜態網站（部落格、Landing Page）。
Serverless Functions (Vercel / AWS Lambda)	✅（平台提供）	✅	✅	✅	拆分成獨立 Functions，自動擴展。適合流量不穩的專案。
Edge Runtime（`runtime: 'edge'`）	❌（Web API 而非 Node）	⚠ 部分支援（僅 Server Components，可撈 API）	❌	⚠ 部分支援（需 Web API，不支援 Node drivers）	全球低延遲運算、Middleware、A/B 測試。
Custom Server (Express/Koa/Fastify + Next)	✅	✅	✅	✅	自訂路由/中介層，整合既有 Node 架構。

補充說明

- **Static Export**：因為沒有 Node 伺服器 → **無法即時渲染**（SSR）、**無法再生**（ISR）、**無 API Routes**。
- **Edge Runtime**：
  - 可以做 SSR（例如 Server Components 即時抓資料），但**無 Node API**（不能用 `fs`、原生 DB drivers）。
  - 適合處理 Header、Cookie、權限驗證、低延遲資料讀取。
- **Serverless Functions**：Vercel/AWS 會把 SSR/ISR/API Route 拆成獨立 Function → 自動擴展，但要注意冷啟動與 DB 連線池。

總結

- **要完整功能（SSR/ISR/API Routes）** → 用 **Node Server** 或 **Standalone**。
- **要純靜態內容** → 用 **Static Export**。
- **要全球低延遲** → 用 **Edge Runtime**（但功能有限制）。
- **要免維運，自動擴展** → 用 **Serverless Functions**（Vercel、AWS Lambda）。

修訂版本 #3
★由 treeman 建立於 17 E 2025 10:27:58
由 treeman 更新於 17 E 2025 10:42:43