READ MODE · ANSWER FIRST

How Netlify Cloud Works

An interactive, visual explanation of How Netlify Cloud Works, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

The whole story in 6 lines

How Netlify Cloud Works becomes easier to reason about when every stage is connected as one system.

1. Git pushes, deploy hooks, and manual drops all converge into the same deploy queue once Netlify has identified the site and branch context.
2. Static files, serverless functions, edge handlers, and manifest metadata are all produced together so the deploy can stay internally...
3. Netlify can reuse unchanged files across deploys, but every successful deploy still gets its own immutable deploy ID and URL space.
4. A POP can fetch a missing file from the immutable deploy store once, warm its cache, and then answer nearby requests directly from the edge.
5. Edge code is for fast request steering and personalization, while Netlify Functions handle the heavier compute and data work.
6. Because production traffic is just an alias to one deploy ID, promotion and rollback are both pointer updates instead of rebuilds.

STAGE 1 OF 8

Setup

Welcome. Netlify is a cloud platform that turns your source code into a live website. Think of it like a factory: code goes in one end, and a fast, globally available site comes out the other. Let us start by learning the vocabulary.

A deploy is a complete snapshot of your site at one moment in time. Every time you push code, Netlify creates a new deploy. It is like saving a new version of a document, except the old versions never disappear.

A CDN (Content Delivery Network) is a group of servers spread across the world. Instead of every visitor waiting for one faraway server, the CDN serves your files from whichever server is closest to them. Think of it like having copies of your homework at every school in the district.

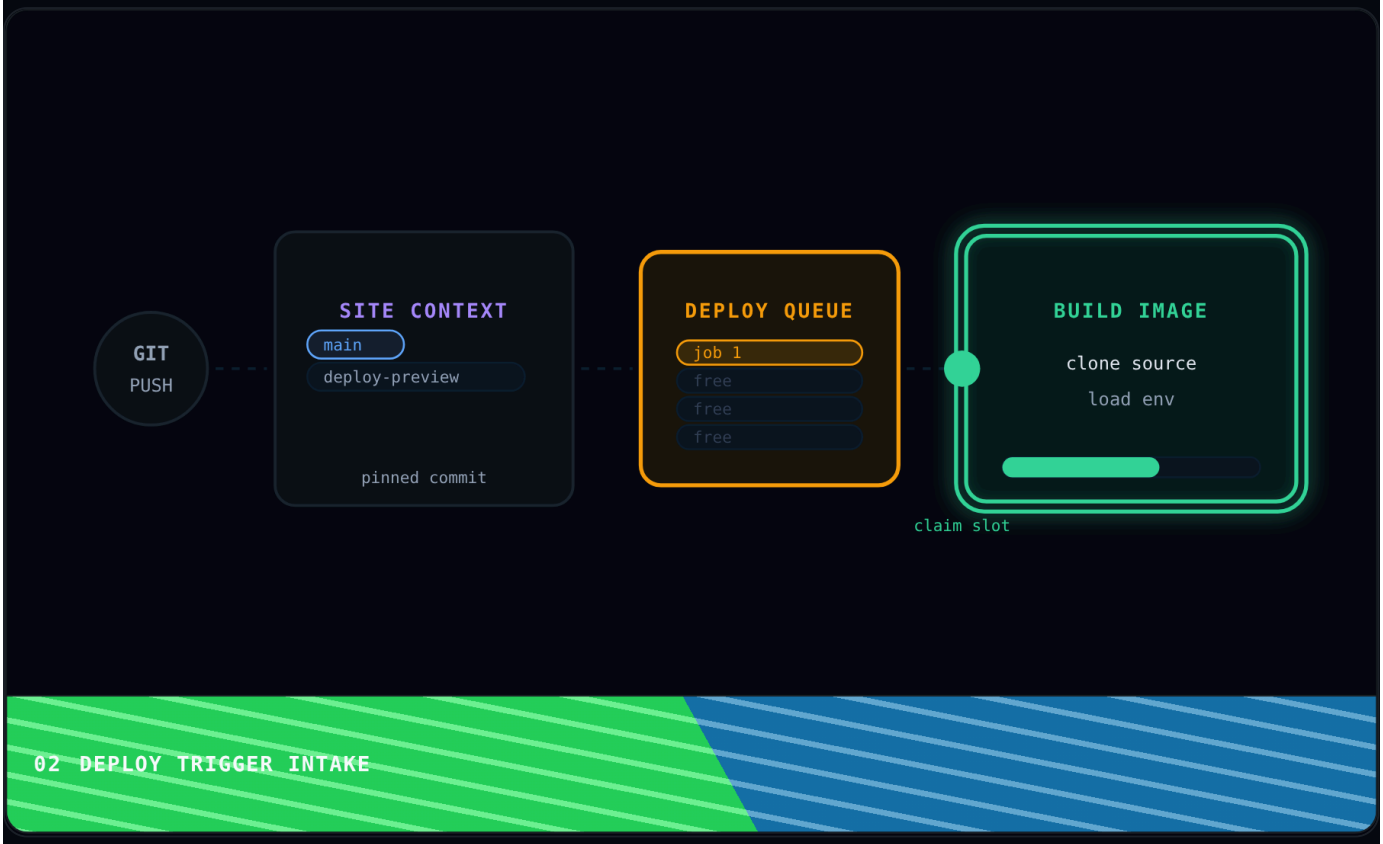
An edge function is a small piece of code that runs at the nearest CDN server, right next to your visitor. It can personalize pages or check permissions before the heavier work begins. A build is the step that compiles your source code into the files a browser can actually use.

Immutable means "cannot be changed." Once Netlify creates a deploy, that snapshot is frozen forever. Nobody can edit it in place. An atomic swap means the live site switches from one deploy to another in a single instant, with no in-between state where half the old site and half the new site are showing.

Over the next six stages we will trace the full journey: how a trigger starts a deploy, how the build assembles output, how deploys get permanent addresses, how static files reach the edge, how dynamic logic runs, and how releases happen safely. Let us begin.

STAGE 2 OF 8

Deploy Trigger Intake



02 DEPLOY TRIGGER INTAKE

Now that we know the vocabulary, let us see where every deploy begins. A trigger enters Netlify when you push code, fire a deploy hook, or drag files into the dashboard. Use the Source control to switch between Git push, Deploy hook, and Manual drop and watch the trigger icon change.

Netlify maps that trigger onto a specific site and branch context. If you increase the Branches stepper, you can see additional branch rows appear inside the site context panel.

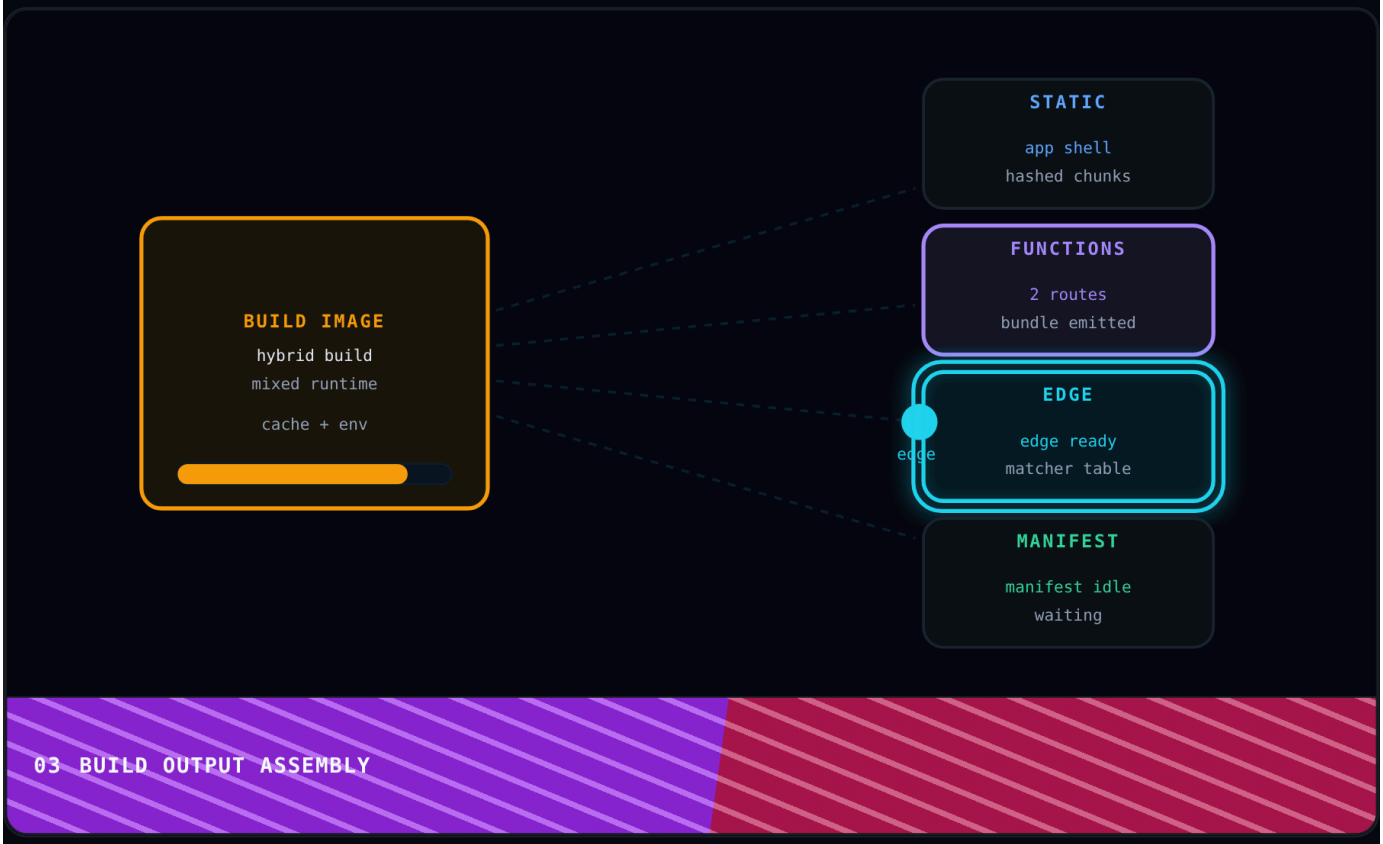
Once the context is resolved, Netlify normalizes the trigger into a single deploy job and places it in a queue. Raising the Queue stepper adds jobs ahead of yours so you can see how the queue fills up.

A build image claims the next job from the queue and pulls the exact source snapshot. This means Netlify locks the commit so nothing can change under the build while it runs.

The build environment is now ready to produce deploy artifacts. Every trigger, whether it came from Git, a webhook, or a manual upload, follows the same path from this point forward. Next we will see what the build actually produces.

STAGE 3 OF 8

Build Output Assembly



Now that the trigger has claimed a build slot, the build image boots up. It restores cached dependencies and loads your environment variables so the compile step starts fast.

The build command compiles your source files into static assets like HTML, CSS, and JavaScript. Switch the App mode control between Static, Hybrid, and SSR to see how the output mix changes inside the build panel.

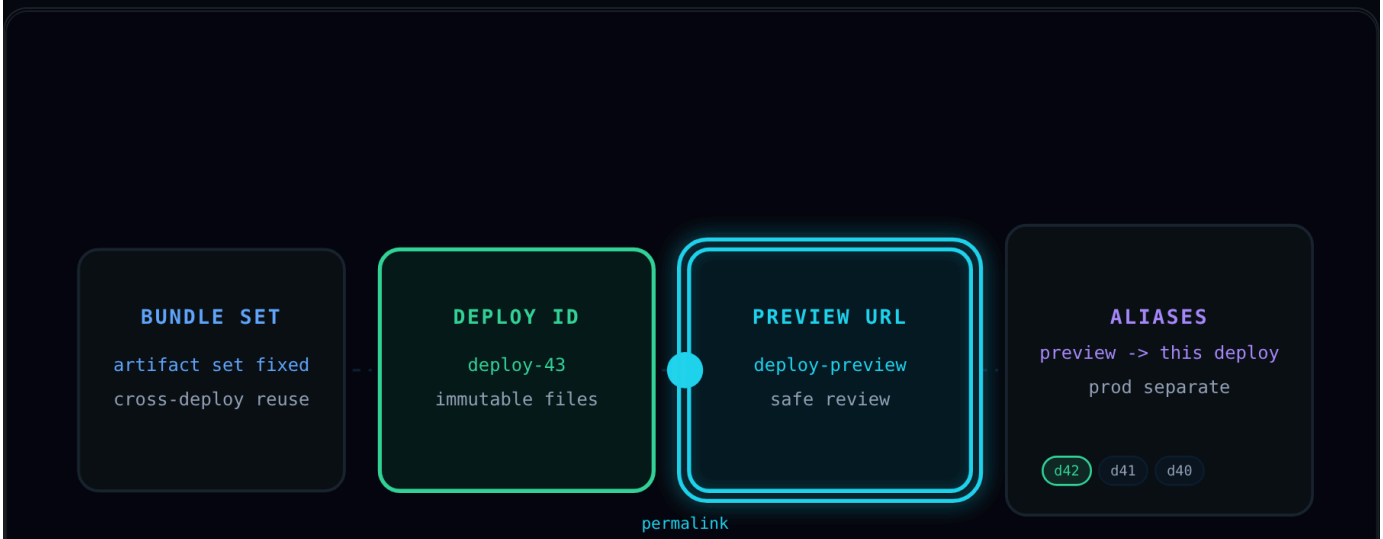
Serverless functions are bundled separately from the static files. You can raise or lower the Functions stepper to add or remove function routes and watch the function panel update.

If your site uses edge handlers, they are compiled in this step. Toggle the Edge control off to see the edge panel dim and the build skip that compile entirely.

Finally, one deploy manifest records every artifact the build produced: static files, functions, edge code, headers, and redirects. This manifest is the single source of truth that the next stage will freeze into an immutable deploy.

STAGE 4 OF 8

Immutable Deploy Addressing



04 IMMUTABLE DEPLOY ADDRESSING

Now that the build has produced its manifest, Netlify fingerprints every file in the bundle. Each file gets a content hash so the platform knows exactly what changed since the last deploy.

Netlify uploads only the files that actually changed and reuses duplicates from earlier deploys. Toggle the Reuse files control off to see what happens when every file is uploaded fresh instead of deduplicated.

A new deploy ID freezes the exact manifest. From this moment on, that deploy is immutable. Nobody can edit its files in place, just like a published book edition that can never be revised.

Every deploy gets a stable permalink. Use the Context control to switch between Preview, Branch, and Prod to see how different URL families point at the same immutable deploy.

Older deploys stay available alongside the new one. Increase the History stepper to see more past deploys in the alias panel. Because every deploy is immutable, you can promote or inspect any of them at any time. Next we will see how these frozen files reach visitors around the world.

STAGE 5 OF 8

Global Static Delivery

05 GLOBAL STATIC DELIVERY

Now that we have an immutable deploy with stable addresses, let us follow a real request. A browser asks for one hashed file from the published deploy. Use the Asset control to switch between JS, Image, and HTML to see different file types flow through the pipeline.

The request lands at the nearest Netlify edge POP, which is a server close to the visitor. The POP checks its local cache to see if it already has that file. Toggle the Warm cache control on to see what happens when the file is already cached.

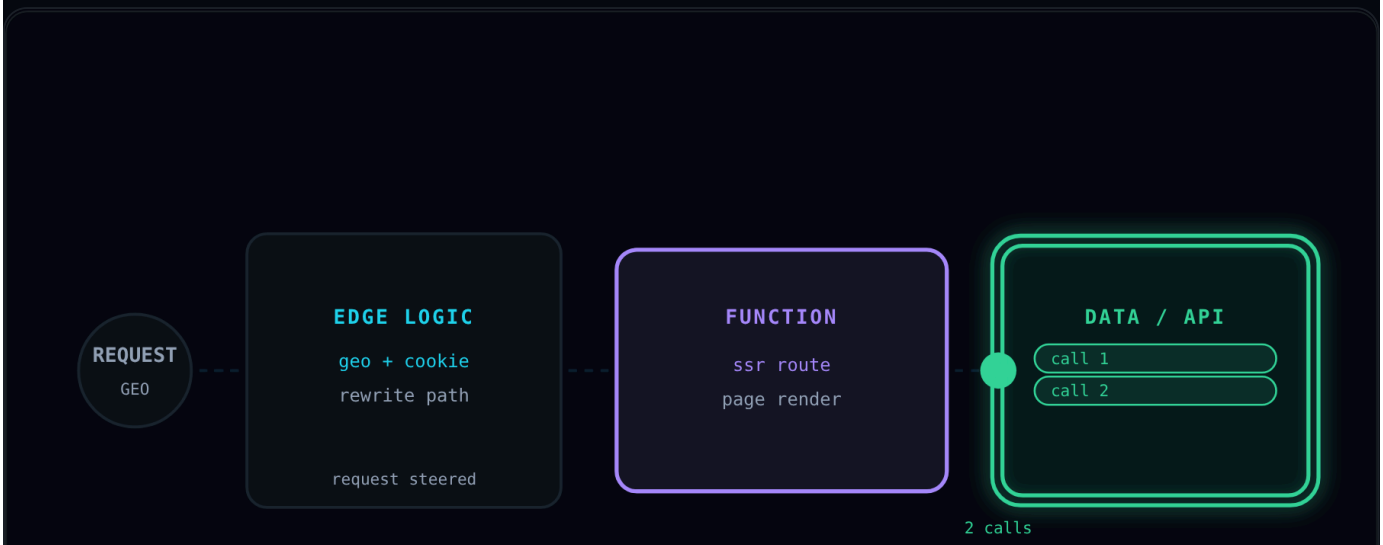
On a cache miss, the POP reaches back to the immutable deploy store and fetches the exact file by its content hash. Because the deploy is frozen, this file can never change unexpectedly.

The file streams back to the POP, which caches it locally and sends it to the visitor at the same time. You can see the cache meter fill up as the edge warms.

The next visitor nearby gets that same file directly from the POP with no round trip to the deploy store. Increase the POPs stepper to see more edge locations around the world. This is how static delivery stays fast everywhere. Next we will look at what happens when a request needs more than a static file.

STAGE 6 OF 8

Edge Logic + Functions



06 EDGE LOGIC + FUNCTIONS

Now that we understand how static files reach the edge, let us see what happens when a request needs dynamic logic. A route that matches an edge handler or serverless function enters the runtime pipeline instead of the static cache. Switch the Request control between Geo page, Auth page, and API to see different dynamic paths.

Edge code runs at the nearest POP, just like the static cache from the previous stage. It can inspect cookies, headers, and visitor geography to rewrite or redirect the request before heavier compute starts. Toggle the Rewrite control off to see the edge pass the request through without changes.

When the request needs heavier compute, Netlify invokes a serverless function. This function handles tasks like rendering a full page, checking authentication, or processing an API call.

The function can call databases or third party APIs to gather the data it needs. Increase the Calls stepper to see the function make more upstream calls inside the data panel.

The finished response flows back through the edge, which attaches final headers and a cache policy. Edge logic handles the fast, lightweight work while functions handle the heavy lifting. Next we will see how Netlify safely promotes a deploy to production.

STAGE 7 OF 8

Atomic Promotion + Rollback

07 ATOMIC PROMOTION + ROLLBACK

Now that we have seen how requests flow through the edge and functions, let us look at how a new deploy goes live. A candidate deploy can sit in preview while you validate it, without disturbing the current production site. Switch the Mode control to Preview to see a deploy that stays in preview without promoting.

Publishing flips the production alias so it points at the new immutable deploy ID. This is an atomic swap: one moment the alias points to the old deploy, and the next moment it points to the new one. There is no in between state. Toggle the Auto publish control off to see what happens when the alias stays locked.

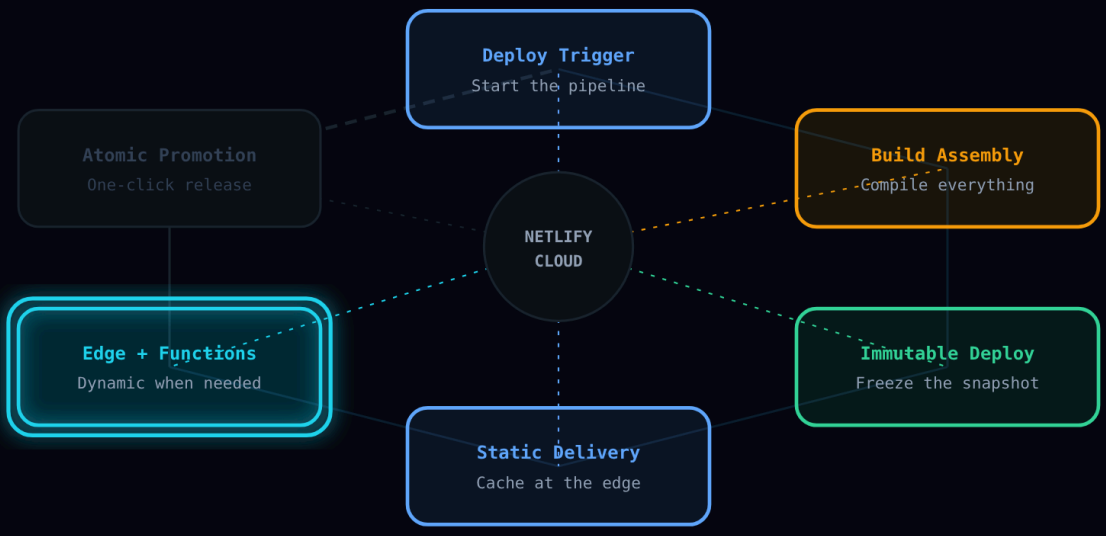
Every edge POP around the world resolves production traffic through that same alias. Because the POP reads the alias pointer, the switch is instant everywhere. Increase the POPs stepper to see more edge locations pick up the change.

The previous deploy is still fully preserved. Remember how we said deploys are immutable? That means the old deploy is still sitting right there, ready to serve traffic again if you need it.

A rollback simply repoints the production alias back to the old deploy ID, and every POP follows that decision immediately. No rebuild, no reupload, just a pointer swap. Switch the Mode control to Rollback to see this flow in action. That is the full pipeline, from trigger to rollback. Let us wrap up.

STAGE 8 OF 8

Recap



08 RECAP

Remember the deploy trigger from Stage 1? That is where every cycle begins. A git push, a deploy hook, or a manual upload kicks off the whole pipeline, just like dropping a letter into a mailbox starts the postal system.

The build step (Stage 2) takes your source code and compiles it into static files, functions, and edge handlers. Think of it as the kitchen in a restaurant: raw ingredients go in, finished dishes come out.

Once built, the output becomes an immutable deploy (Stage 3). It gets a permanent ID and a permanent URL. Like a published book edition, it can never be altered after the fact.

Static files are then pushed to CDN servers around the world (Stage 4). The first visitor in each region warms the cache, and everyone after that gets near-instant responses from a server right around the corner.

When a request needs more than a static file, edge functions and serverless functions step in (Stage 5). Edge code handles lightweight personalization close to the visitor, while heavier compute runs in a serverless function.

Finally, the atomic promotion step (Stage 6) flips the live site from one deploy to another in a single pointer swap. If something goes wrong, rollback does the same thing in reverse, with no rebuild needed.

And the loop closes. The next git push starts a brand new cycle through all six stages. Every deploy is immutable, every release is atomic, and the edge serves traffic globally. That is how Netlify Cloud works.