LECTURE 32

Domain, DNS and Deployment

The last step: taking your MERN app off your laptop and putting it on the Internet.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Domain names, registrars, and the main DNS record types
2. How DNS resolution actually works, and what TTL means
3. Hosting options: shared hosting, VPS, PaaS, and serverless
4. Deploying a React frontend and Express backend, plus reverse proxies and monitoring

A Human-Readable Stand-In for an IP

1. A domain name (example.com) stands in for a numeric IP address (93.184.216.34) computers actually use
2. Read right to left: .com is the top-level domain (TLD); example is the second-level domain you register
3. www (or any prefix) is a subdomain — used to organize services, e.g. api.example.com, blog.example.com

Buying a Domain

1. You register through a registrar — a company accredited to sell domains, e.g. Namecheap, GoDaddy, Google Domains
2. Registrars reserve the name for you (usually one year at a time, renewable)
3. They do NOT host your website's content — that's a separate job

Domains and Hosting Are Two Separate Purchases

NOTE

A very common beginner confusion: buying a domain does not, by itself, put a website online. You still need a server somewhere to store and run your application, and you need to tell your domain where that server is — which is exactly what DNS does.

The Internet's Phone Book

1. DNS (Domain Name System) translates human-readable domain names into IP addresses
2. You look up a name, DNS gives you a number
3. DNS information is organized into records, stored on nameservers, grouped into zones (one zone per domain)

The Record Types You'll Use Most

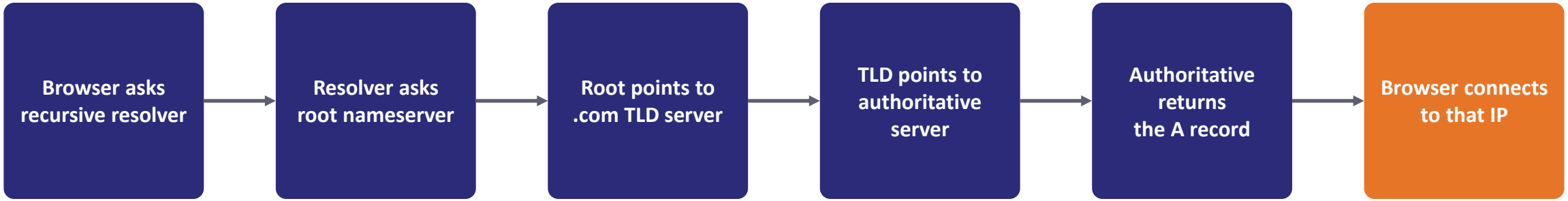
Record	Purpose	Example
A	Maps a domain/subdomain to an IPv4 address	example.com -> 93.184.216.34
AAAA	Maps a domain/subdomain to an IPv6 address	example.com -> 2606:2800:...
CNAME	Maps a domain/subdomain to another domain name	www.example.com -> example.com
MX	Which mail servers handle email, and priority	example.com -> mail.example.com
TXT	Arbitrary text — ownership verification, SPF/DKIM	"v=spf1 include:..."

CNAME Has a Root-Domain Limit

TIP

Hosting platforms like Vercel/Netlify often ask you to point a subdomain at THEIR domain via CNAME, since their own IPs can change. But you generally can't put a CNAME on the bare root domain alongside records like MX — a historical DNS rule. Most platforms offer a workaround ("ALIAS"/"ANAME" records) for pointing a root domain at them.

What Happens When You Type a URL



The recursive resolver (your ISP, or a public one like 8.8.8.8) does this legwork on your behalf. The authoritative nameserver — configured at your registrar — holds the actual, official records for your domain.

Time to Live

1. Every DNS record has a TTL (in seconds) — how long resolvers may cache that answer before asking again

2. A TTL of 3600 means "valid for one hour; don't re-check more often than that"

A Record With Its TTL

```
example.com.    3600    IN    A    93.184.216.34
```

The Trade-Off

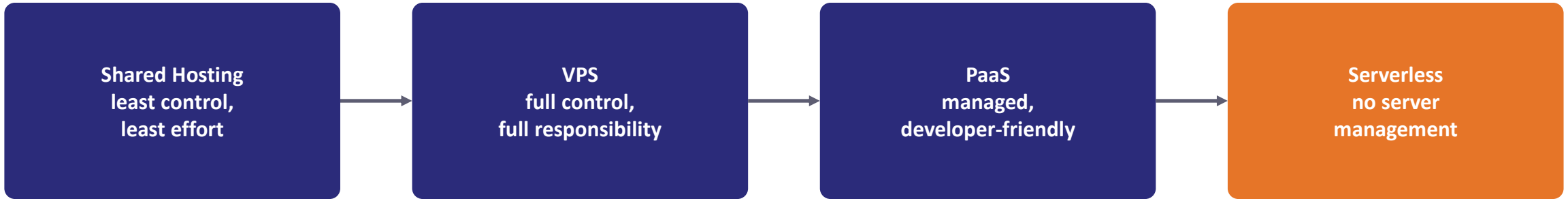
TTL	Effect
High (e.g. 24h)	Less nameserver load, faster repeated lookups — but changes take longer to reach everyone
Low (e.g. 5 min)	Changes propagate fast — at the cost of more frequent lookups

Lower It Before You Move Hosts

TIP

If you know you're about to change a DNS record — e.g. moving your site to a new host — lower the TTL a day or two in advance, so resolvers pick up the new value quickly instead of serving a stale answer for hours.

From Least to Most Managed



Four Ways to Run Your App

SHARED HOSTING

- Your site shares a server with many others
- Cheap, simple — often just FTP
- Poor fit for a Node.js + MongoDB app

VPS

- Your own isolated slice, root access
- DigitalOcean, Linode, AWS EC2
- You configure and secure everything

PAAS

- Platform provisions servers for you
- Render, Railway, Heroku
- The sweet spot for student full-stack projects

SERVERLESS

- Code runs only per request, scales to zero
- AWS Lambda, Vercel/Netlify functions
- Pay per use; no long-lived DB connections

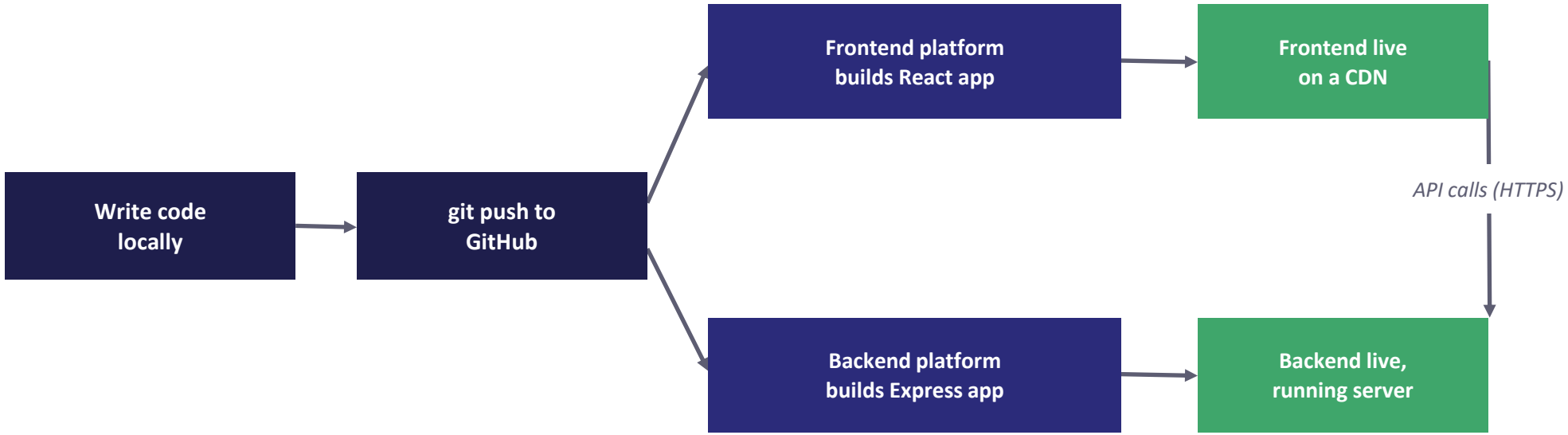
Comparing the Four

Option	Control	Setup Effort	Good For
Shared hosting	Low	Very low	Simple static/PHP sites
VPS	Full	High	Custom server setups, learning ops
PaaS	Medium	Low	Most student/startup full-stack apps
Serverless	Low (by design)	Low, different model	Spiky, event-driven traffic

One Repo, Two Deploy Targets

1. A typical MERN project deploys its React frontend and Express backend SEPARATELY
2. Each goes to a platform suited for it — a static-file host for the frontend, a PaaS for the backend
3. Both platforms connect to the same GitHub repository and rebuild automatically

From git push to Live



Every push to the connected branch triggers an automatic rebuild and redeploy on both platforms — continuous deployment.

Development vs. Production

```
# In production, compile React into a small set of
# static, optimized HTML/CSS/JS files
npm run build
```

The dev server (npm start) is heavy and unminified, meant for a fast feedback loop while coding — not for real users.

Production Environment Variables

```
# .env.example -- committed as documentation, not real secrets
DATABASE_URL=your-production-mongodb-uri
JWT_SECRET=your-jwt-secret
NODE_ENV=production
PORT=5000
```

Real values go into your hosting platform's dashboard — the same .env-based approach from Lecture 30, but for production.

e.g. Vercel or Netlify

1. Push your React project to a GitHub repository
2. Connect that repository to Vercel/Netlify through their dashboard
3. Configure the build command (npm run build) and output directory (build or dist)
4. The platform builds and deploys, giving you a live URL immediately
5. Optionally connect your own domain via the CNAME record it tells you to add

e.g. Render

1. Push your Express project to GitHub
2. Connect the repository to Render and create a new "Web Service"
3. Configure the start command (node server.js) and environment variables in the dashboard
4. Render builds and starts your server, with HTTPS already configured
5. Update your deployed frontend to point its API requests at this live backend URL

Free Tiers Spin Down

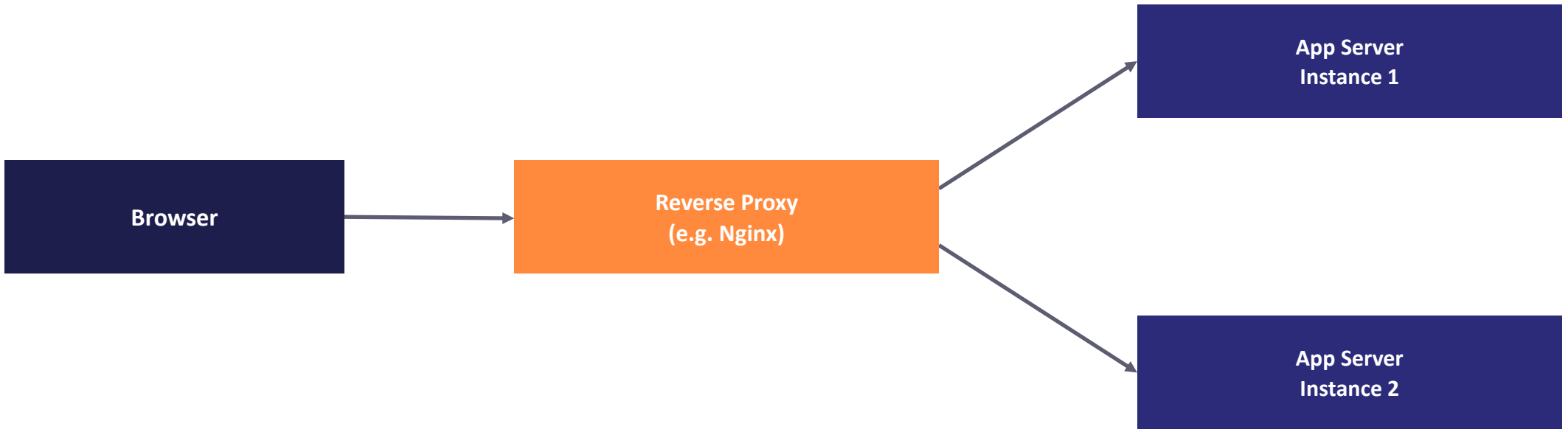
WARNING

Free tiers on platforms like Render often "spin down" your backend after inactivity to save resources, then take a few seconds to "wake up" on the next request. Normal for a student project — paid tiers keep your service running continuously.

The Server in Front of Your Server

1. A reverse proxy sits in front of your actual application server, receiving requests first and forwarding them
2. Most hosting platforms run one for you, even if you never interact with it directly
3. On a VPS you'd configure one yourself, commonly with software like Nginx

One Front Door, Many Backends



One proxy commonly handles three jobs at once: terminating SSL/TLS (one place manages the certificate), load balancing across app instances, and serving static files directly.

Knowing It's Actually Working

UPTIME MONITORING

- A service periodically pings your app
- Alerts you if it stops responding
- e.g. UptimeRobot, or built into many PaaS dashboards

LOGS

- console.log and error output from your server
- See what happened around a problem
- Most platforms give a live logs dashboard

ERROR TRACKING

- Tools like Sentry capture unhandled exceptions
- Stack trace + which user/request triggered it
- Automatic, no need to reproduce manually

MONITORING

Check Your Logs Right After Deploying

TIP

Even for a class project, check your hosting platform's logs dashboard right after deploying, and again the next day. It's the fastest way to catch a crash, a missing environment variable, or a database connection issue that isn't obvious from clicking around the live site.

KEY TAKEAWAYS

Congratulations — You've Reached the End of CSC336

1. A domain name is a human-readable stand-in for an IP address, purchased through a registrar — buying one doesn't host your app by itself.
2. DNS translates names to IPs via A, AAAA, CNAME, MX, and TXT records; TTL controls how long each answer is cached along the way.
3. Hosting ranges from shared hosting to VPS to PaaS (the sweet spot for most student MERN apps) to serverless.
4. Deploying a MERN app means a build step for the React frontend and a continuously running Express server, each with production environment variables.
5. A reverse proxy handles SSL termination, load balancing, and static files — most PaaS platforms run one for you automatically.
6. You went from how the web works to building, securing, and deploying a full-stack MERN app — that foundation carries into any framework you use next. Advanced Web Technologies (CSC337) picks up from here.