01 / The finished system

Deploy a React single-page application, a Node.js API and a MySQL database on one fresh Ubuntu 24.04 LTS VPS. Nginx serves the compiled frontend, forwards API calls and terminates HTTPS. This is a public, read-only demonstration app, not an authentication or payments system.

Traffic and files

```
Browser -> HTTPS :443 -> Nginx
  /             -> /var/www/codekits-demo (React build)
  /api/notes    -> 127.0.0.1:3000 (Node.js)
  Node.js       -> 127.0.0.1:3306 (MySQL)

Source project:  ~/codekits-tutorial/project
API service:     codekits-api.service
Runtime secret:  /etc/codekits-demo.env (root, mode 600)
Database:        codekits_demo
App DB account:  webapp@127.0.0.1 (SELECT only)
```

LOCAL means your own computer. SERVER means your SSH session on the VPS. DNS means your domain provider dashboard.

Replace YOUR_SERVER_IP with your VPS public address and demo.example.com with a domain you control. The example domain is not a deployment target.

The video is a captioned command walkthrough. Displayed results are expectations, not a claim that a live VPS or certificate was provisioned during video production.

02 / Provision the VPS and set DNS

Use a fresh Ubuntu 24.04 LTS server with sudo access, a public IP and an x64 or ARM64 CPU. Around 2 GB RAM gives this small example more room to install and build dependencies. Resource needs vary with traffic and application size.

DNS dashboard - example for demo.yourdomain.com

```
Type     Host       Value
A        demo       YOUR_SERVER_IP

Optional AAAA: only add your working server IPv6.
Do not leave an old AAAA record pointing elsewhere.
```

Add your SSH public key when creating the VPS. Keep the private key on your computer.

In the provider firewall, allow your SSH port (22 in this tutorial), TCP 80 and TCP 443. Restrict SSH to your IP where practical. Do not expose 3000 or 3306.

Use DNS-only routing while setting up this direct-origin example. If you use a CDN proxy, certificate and proxy settings need separate consideration.

An A record maps a name to IPv4. A CNAME can point www to the main name, but only request certificates for names that already resolve correctly.

03 / SSH access and a deployment user

LOCAL: use an existing key or create a dedicated key below. Do not overwrite an existing key file. Add the .pub file to the provider; then compare the first SSH host-key fingerprint with the provider console. Some providers use ubuntu rather than root for the first login.

LOCAL - key and first connection

```
ssh-keygen -t ed25519 -f ~/.ssh/codekits_vps -C codekits-vps
ssh -i ~/.ssh/codekits_vps root@YOUR_SERVER_IP
```

SERVER - as the initial admin

```
sudo adduser deploy
sudo usermod -aG sudo deploy
sudo install -d -m 700 -o deploy -g deploy /home/deploy/.ssh
sudo install -m 600 -o deploy -g deploy \
  "$HOME/.ssh/authorized_keys" /home/deploy/.ssh/authorized_keys
```

LOCAL - new terminal, test deploy access

```
ssh -i ~/.ssh/codekits_vps deploy@YOUR_SERVER_IP
sudo -v
```

The key-copy command assumes the initial account already has an authorized_keys file from provider SSH-key setup. If it does not, add your public key through the provider before proceeding.

Keep the initial SSH session open until the deploy login and sudo both work. Subsequent SERVER commands use the deploy account. This lesson keeps the current SSH authentication configuration.

04 / Install packages and allow traffic

SERVER: apply updates and install the web server, database and utilities. Complete any package prompts. If the system requests a reboot, reboot and reconnect before continuing.

SERVER - base packages

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y nginx mysql-server ufw curl \
  ca-certificates xz-utils unzip rsync openssl dnsutils snapd
sudo systemctl enable --now nginx mysql
```

SERVER - firewall, with SSH allowed first

```
sudo ufw allow OpenSSH
sudo ufw allow 'Nginx Full'
sudo ufw enable
sudo ufw status verbose
```

OpenSSH permits the default SSH port. If your provider uses a custom SSH port, allow that port before enabling UFW.

Nginx Full permits HTTP and HTTPS. The provider firewall and the operating-system firewall must both permit the traffic.

Open a second deploy SSH session to verify access still works before closing the first one.

05 / Upload the included working project

Download the starter ZIP, then open a local terminal in the folder containing it. The package includes all React and API source, exact dependency lockfiles, SQL, deployment configurations and readable helper scripts. No GitHub repository or paid template is required.

LOCAL - upload

```
scp -i ~/.ssh/codekits_vps \
  CodeKits-Ubuntu-VPS-Starter-Kit.zip deploy@YOUR_SERVER_IP:~/
```

SERVER - extract and enter the project

```
mkdir -p ~/codekits-tutorial
unzip ~/CodeKits-Ubuntu-VPS-Starter-Kit.zip -d ~/codekits-tutorial
cd ~/codekits-tutorial/project
```

Project layout

```
api/          Express API, MySQL pool and HTTP tests
frontend/     React source and Vite build configuration
database/     Schema and public demonstration rows
deploy/       Nginx server block and systemd unit
scripts/      Node install, database setup, deploy, backup
```

Unless a section says LOCAL, run the remaining commands in the SSH session. Return to ~/codekits-tutorial/project when using a helper script.

These helpers target dedicated tutorial directories. Use a fresh server for this exercise rather than mixing it into an existing deployment.

06 / Install a patched Node.js 24 release

SERVER, from the project directory: run bash scripts/install-node24.sh. The script downloads the current patch release in the Node 24 LTS line, chooses x64 or ARM64 and checks the archive against the published SHA-256 list. This checksum checks file integrity; signed-release verification is documented by Node.js.

Included script: scripts/install-node24.sh

```
#!/usr/bin/env bash
set -euo pipefail
# Fresh Ubuntu server; download the current patched release in the Node 24 line.
case "$(uname -m)" in
  x86_64) node_arch=x64 ;;
  aarch64|arm64) node_arch=arm64 ;;
  *) echo 'This tutorial supports x64 and ARM64.' >&2; exit 1 ;;
esac
node_tmp="$(mktemp -d)"
trap 'rm -rf "$node_tmp"' EXIT
cd "$node_tmp"
node_base='https://nodejs.org/dist/latest-v24.x'
curl -fsSLo SHASUMS256.txt "$node_base/SHASUMS256.txt"
node_archive="$(awk -v arch="$node_arch" \
  '$2 ~ ("^node-v24\\.[0-9]+\\.[0-9]+-linux-" arch "\\.tar\\.xz$") {print $2; exit}' \
  SHASUMS256.txt)"
test -n "$node_archive"
curl -fsSLO "$node_base/$node_archive"
awk -v file="$node_archive" '$2 == file' SHASUMS256.txt | sha256sum -c -
sudo install -d -m 755 /opt/node24
sudo tar -xJf "$node_archive" -C /opt/node24 --strip-components=1
sudo ln -sfn /opt/node24/bin/node /usr/local/bin/node
sudo ln -sfn /opt/node24/bin/npm /usr/local/bin/npm
sudo ln -sfn /opt/node24/bin/npx /usr/local/bin/npx
/opt/node24/bin/node --version
/usr/local/bin/npm --version
```

The application service uses /opt/node24/bin/node directly, so it does not depend on an interactive shell or nvm.

This installation is outside apt. Re-run the installer during planned maintenance to apply Node 24 patches, then restart and check the API. Review major-version upgrades separately.

07 / Create the database and server secret

SERVER: run bash scripts/init-database.sh once on the fresh installation. It binds MySQL to loopback, creates the database and its SELECT-only account, loads the sample rows and writes a root-only environment file. Do not run this initializer again to update the application.

Included script: scripts/init-database.sh

```
#!/usr/bin/env bash
set -euo pipefail
project_root="$(cd -- "$(dirname -- "${BASH_SOURCE[0]}")/.." && pwd)"
sudo -v
if sudo test -e /etc/codekits-demo.env; then
  echo 'Environment file already exists. Stop: do not reset a deployed database.' >&2
  exit 1
fi
sudo install -m 644 /dev/null /etc/mysql/mysql.conf.d/99-codekits-loopback.cnf
sudo tee /etc/mysql/mysql.conf.d/99-codekits-loopback.cnf >/dev/null <<'CONF'
[mysqld]
bind-address = 127.0.0.1
mysqlx-bind-address = 127.0.0.1
CONF
sudo systemctl restart mysql

# Hex avoids shell/SQL quoting problems. Never put this secret in React.
DB_PASSWORD="$(openssl rand -hex 32)"
sudo mysql <<SQL
CREATE DATABASE codekits_demo CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'webapp'@'127.0.0.1' IDENTIFIED BY '${DB_PASSWORD}';
GRANT SELECT ON codekits_demo.* TO 'webapp'@'127.0.0.1';
SQL
sudo mysql codekits_demo < "$project_root/database/schema.sql"
sudo install -m 600 /dev/null /etc/codekits-demo.env
printf 'NODE_ENV=production\nPORT=3000\nDB_HOST=127.0.0.1\nDB_USER=webapp\nDB_NAME=codekits_demo\nDB_PASSWORD=%s\n' \
  "$DB_PASSWORD" | sudo tee /etc/codekits-demo.env >/dev/null
unset DB_PASSWORD
echo 'Created the database, read-only app account and root-only environment file.'
```

The hex password is generated on your server. It is never embedded in the frontend, copied into the video or committed to the project.

The script refuses to run if its environment file already exists. If a first run fails part-way, inspect the database state and logs before retrying.

08 / Seed data and connect Node to MySQL

The three demonstration notes originate in MySQL. Node reads credentials from its process environment, opens a small connection pool and checks database access before listening on 127.0.0.1:3000. The full server file also validates configuration and closes the pool on shutdown.

database/schema.sql

```
CREATE TABLE notes (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  body VARCHAR(200) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO notes (body) VALUES
  ('Hello from MySQL'),
  ('Node.js connects the pieces'),
  ('Nginx brings it online');
```

Connection excerpt from api/server.mjs

```
const pool = mysql.createPool({
  host: process.env.DB_HOST,
  port: 3306,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  waitForConnections: true,
  connectionLimit: 5,
  queueLimit: 50,
  connectTimeout: 5000
});

// Refuse to start if the credentials or database are wrong.
await pool.query('SELECT 1');
const server = createApp(pool).listen(port, '127.0.0.1', () => {
  console.log(`API listening on 127.0.0.1:${port}`);
});
```

The database account has no permission to create tables or change notes. Administrative migrations use a separate administrative connection.

Use parameter placeholders for queries with user input. This sample has fixed read queries and no public write routes.

09 / The API routes

Express returns JSON under /api. Its health check actually queries MySQL, so it can detect a database problem as well as a stopped Node process. The browser receives a generic failure message while the server logs an error code.

api/app.mjs - complete route implementation

```
import express from 'express';

// Inject the pool so HTTP behaviour can be tested without a live database.
export function createApp(pool) {
  const app = express();
  app.disable('x-powered-by');
  app.set('trust proxy', 'loopback');
  app.use((_req, res, next) => {
    res.set('Cache-Control', 'no-store');
    next();
  });

  app.get('/api/health', async (_req, res) => {
    await pool.query('SELECT 1');
    res.json({ ok: true });
  });

  app.get('/api/notes', async (_req, res) => {
    const [rows] = await pool.execute(
      'SELECT id, body FROM notes ORDER BY id LIMIT 20'
    );
    res.json({ notes: rows });
  });

  app.use((_req, res) => res.status(404).json({ error: 'Not found' }));
  app.use((error, _req, res, _next) => {
    // Log the error code, not credentials or a full connection object.
    console.error('Request failed:', error.code || error.name);
    res.status(503).json({ error: 'Service temporarily unavailable' });
  });
  return app;
}
```

Run npm --prefix api test after installing its dependencies to exercise the HTTP behaviour with a test database adapter.

Unknown routes and POST requests return 404. The sample exposes only public demonstration text.

10 / Connect the React frontend

The included frontend/src/main.jsx fetches /api/notes and displays each returned row. This relative URL uses the same domain and HTTPS connection as the page. No database password belongs in React source or any VITE_ variable.

Key excerpts - full component is in the project

```
const response = await fetch('/api/notes');
if (!response.ok) throw new Error('API unavailable');
const data = await response.json();
setNotes(data.notes);

// Render the rows:
{notes.map(note => (
  <article key={note.id}>
    <h2>{note.body}</h2>
  </article>
))}
```

Build commands used by the deployment script

```
npm --prefix frontend ci
npm --prefix frontend run build
# Output: frontend/dist/
```

This is a Vite-built React single-page app. Nginx serves its dist files. The Vite development server is not part of the production deployment.

Nginx forwards /api calls to Node, so this example does not require a separate API domain or browser CORS configuration.

The development-only Vite proxy in vite.config.js is separate from the production Nginx proxy.

11 / Build, copy and start the API

SERVER, project directory: run bash scripts/deploy.sh. It installs from the lockfiles, builds React, creates a separate service user, copies files and starts the API under systemd. Dependencies are installed as deploy; the running service has no sudo privileges.

Included script: scripts/deploy.sh

```
#!/usr/bin/env bash
set -euo pipefail
project_root="$(cd -- "$(dirname -- "${BASH_SOURCE[0]}")/.." && pwd)"
cd "$project_root"
sudo -v
test -x /opt/node24/bin/node
sudo test -f /etc/codekits-demo.env
# Install as the signed-in deployment user, never using sudo npm.
npm --prefix frontend ci
npm --prefix frontend run build
npm --prefix api ci --omit=dev
if ! id webapp >/dev/null 2>&1; then
  sudo useradd --system --user-group --no-create-home \
    --home-dir /srv/codekits-demo --shell /usr/sbin/nologin webapp
fi
sudo install -d -m 755 /srv/codekits-demo/api /var/www/codekits-demo
# Only these dedicated tutorial directories are replaced. Run after a backup.
sudo rsync -a --delete --chown=root:root --chmod=D755,F644 \
  api/ /srv/codekits-demo/api/
sudo rsync -a --delete --chown=root:root --chmod=D755,F644 \
  frontend/dist/ /var/www/codekits-demo/
sudo install -m 644 deploy/codekits-api.service \
  /etc/systemd/system/codekits-api.service
sudo systemctl daemon-reload
sudo systemctl enable codekits-api
sudo systemctl restart codekits-api
curl --retry 10 --retry-delay 1 --retry-connrefused --fail \
  http://127.0.0.1:3000/api/health
printf '\nDeployment complete. Configure Nginx next if this is the first run.\n'
```

The two rsync commands replace only /srv/codekits-demo/api and /var/www/codekits-demo. On updates, take a backup first. This teaching workflow has a short restart window; it is not a zero-downtime release system.

If the health check fails, stop here and inspect sudo journalctl -u codekits-api -n 80 --no-pager.

12 / Keep Node running with systemd

The deployment script installs this unit and enables it at boot. The root-owned environment file is read by systemd before it starts the process as webapp. Restart=on-failure restarts a failed process; the start limit prevents an endless rapid crash loop.

deploy/codekits-api.service

```
[Unit]
Description=CodeKits demonstration Node.js API
After=network.target mysql.service
Wants=mysql.service
StartLimitIntervalSec=60
StartLimitBurst=10

[Service]
Type=simple
User=webapp
Group=webapp
WorkingDirectory=/srv/codekits-demo/api
EnvironmentFile=/etc/codekits-demo.env
ExecStart=/opt/node24/bin/node /srv/codekits-demo/api/server.mjs
Restart=on-failure
RestartSec=5
TimeoutStopSec=10
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true
UMask=0077

[Install]
WantedBy=multi-user.target
```

SERVER - useful checks

```
sudo systemctl status codekits-api --no-pager
sudo journalctl -u codekits-api -n 80 --no-pager
curl -fsS http://127.0.0.1:3000/api/health
```

Expected health response when MySQL is reachable: {"ok":true}.

After correcting a repeated startup failure, use sudo systemctl reset-failed codekits-api and then sudo systemctl restart codekits-api.

13 / Nginx: frontend and API on one domain

This complete HTTP configuration is installed before Certbot adds HTTPS. The root points only to the React build. The /api/ location proxies to the private Node listener. Keeping proxy_pass without a trailing slash preserves /api/notes at the upstream server.

deploy/nginx.conf

```
# Replace YOUR_DOMAIN with a domain you control, e.g. demo.example.com.
server {
    listen 80;
    listen [::]:80;
    server_name YOUR_DOMAIN;
    root /var/www/codekits-demo;
    index index.html;
    server_tokens off;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /assets/ {
        try_files $uri =404;
        add_header Cache-Control "public, max-age=31536000, immutable";
    }

    location = /index.html {
        add_header Cache-Control "no-cache";
    }

    location /api/ {
        # NO trailing slash: keep /api/notes when forwarding to Express.
        proxy_pass http://127.0.0.1:3000;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_connect_timeout 5s;
        proxy_read_timeout 30s;
    }

    location ~ /\.(?!well-known) {
        deny all;
    }
}
```

try_files supplies index.html for client-side routes. Missing /assets files return a real 404 instead of HTML.

Hashed assets can be cached; index.html is revalidated. The browser receives fresh references after a deployment.

14 / Enable the site and confirm HTTP

SERVER, project directory: set DOMAIN to a real domain or subdomain you control. The variable lasts only in this shell; set it again after reconnecting. Install the HTTP server block, check it and reload Nginx.

SERVER - install the site

```
DOMAIN=demo.example.com  # replace with your domain
sed "s/YOUR_DOMAIN/$DOMAIN/g" deploy/nginx.conf | \
  sudo tee /etc/nginx/sites-available/codekits-demo >/dev/null
sudo ln -sfn /etc/nginx/sites-available/codekits-demo \
  /etc/nginx/sites-enabled/codekits-demo
sudo nginx -t
# Continue only when the syntax check succeeds:
sudo systemctl reload nginx
curl -fsS -H "Host: $DOMAIN" http://127.0.0.1/api/health
```

SERVER - public DNS and HTTP checks

```
dig +short A "$DOMAIN" @1.1.1.1
dig +short AAAA "$DOMAIN" @1.1.1.1
curl -I "http://$DOMAIN"
```

Confirm A resolves to your server. Any AAAA record must also lead to this server with working IPv6. DNS caching can delay changes.

From your own computer or phone, visit http://your-domain and confirm the page and cards work before asking for a certificate.

If the server IP shows the default Nginx page, that does not prove the domain server block is broken. Test the correct Host header or domain.

After HTTPS is configured, do not overwrite the live Nginx configuration with the original HTTP template.

15 / Add HTTPS and test renewal

SERVER: use the Certbot Nginx plugin after public HTTP and DNS work. Enter your email and review the subscriber agreement at the prompts. Certbot obtains a certificate, updates the matching server block and redirects HTTP to HTTPS.

SERVER - certificate and renewal

```
sudo snap install --classic certbot
DOMAIN=demo.example.com  # replace again if needed
sudo /snap/bin/certbot --nginx --redirect -d "$DOMAIN"
sudo nginx -t
curl -I "https://$DOMAIN"
sudo /snap/bin/certbot renew --dry-run
sudo systemctl list-timers --all | grep certbot
```

Only add -d www.your-domain if that name already resolves correctly and appears in your Nginx server_name. One working domain is enough for this lesson.

The snap installation normally provides an automatic renewal timer. Check the timer and the dry run; do not assume renewal works just because first issuance worked.

Keep inbound port 80 available for HTTP-01 renewals, including in the provider firewall. HTTPS uses 443.

Do not edit certificate files or hard-code an expiry interval. Certificates have finite lifetimes; renewal monitoring matters.

16 / Verify the complete integration

SERVER: request the public HTTPS endpoints and inspect the listening sockets. Then change a demonstration row administratively and refresh the page. Seeing the new text confirms the browser, Nginx, Node and database path works together.

SERVER - complete stack checks

```
DOMAIN=demo.example.com
curl -fsS "https://$DOMAIN/api/health"
curl -fsS "https://$DOMAIN/api/notes"
sudo ss -lntp
sudo systemctl is-enabled nginx mysql codekits-api
sudo mysql codekits_demo -e \
  "UPDATE notes SET body='Hello from the live database' WHERE id=1;"
```

Optional fresh-server reboot check

```
sudo reboot
# Reconnect from your own computer:
ssh -i ~/.ssh/codekits_vps deploy@YOUR_SERVER_IP
# On the server:
sudo systemctl is-active nginx mysql codekits-api
curl -fsS http://127.0.0.1:3000/api/health
```

Expect Nginx to listen publicly on 80 and 443; Node on 127.0.0.1:3000; MySQL on 127.0.0.1:3306. Its optional X Protocol port should also be loopback.

In the browser Network panel, /api/notes should return 200 and JSON. The page should use HTTPS without mixed-content errors.

Use the reboot check only when a short interruption to this fresh demo server is acceptable.

17 / Back up and practise a restore

The included backup script uses a consistent InnoDB snapshot and writes a compressed SQL dump with private permissions. Choose the actual generated filename in place of CHOSEN.sql.gz. Copy backups off the VPS; a backup on the same failed disk will not help.

SERVER - backup and isolated restore test

```
cd ~/codekits-tutorial/project
bash scripts/backup.sh "$HOME/backups"
# Validate a selected backup:
gzip -t "$HOME/backups/CHOSEN.sql.gz"

# Test a restore into a NEW database:
sudo mysql -e "CREATE DATABASE codekits_restore;"
gunzip -c "$HOME/backups/CHOSEN.sql.gz" | \
  sudo mysql codekits_restore
sudo mysql codekits_restore -e "SELECT COUNT(*) FROM notes;"
```

LOCAL - copy database backups off the server

```
mkdir -p backups
scp -i ~/.ssh/codekits_vps \
  'deploy@YOUR_SERVER_IP:~/backups/*.sql.gz' ./backups/
```

The dump does not include MySQL user accounts, the runtime password, app source or TLS files. Preserve your source plus an encrypted copy of required configuration and secrets separately.

The restore test creates a separate codekits_restore database. If that CREATE DATABASE command fails, stop; do not restore into an existing database. Never test a restore over live data.

Protect backup files and use an access-controlled destination. Establish a backup schedule appropriate to how often your real data changes.

18 / Update and recover

Before changing a working deployment, back up the database and save a private snapshot of application/configuration files. Upload revised source to your home-directory project, review it, then deploy. Never rerun the database initializer as an update step.

SERVER - private pre-update snapshot

```
sudo install -d -m 700 /var/backups/codekits-app
SNAPSHOT="/var/backups/codekits-app/app-$(date -u +%Y%m%dT%H%M%SZ).tgz"
sudo tar -C / -czf "$SNAPSHOT" \
  srv/codekits-demo/api var/www/codekits-demo \
  etc/systemd/system/codekits-api.service \
  etc/nginx/sites-available/codekits-demo etc/codekits-demo.env
sudo chmod 600 "$SNAPSHOT"
```

SERVER - deploy a reviewed update

```
cd ~/codekits-tutorial/project
# After updating reviewed source and lockfiles:
npm --prefix api ci
npm --prefix api test
bash scripts/deploy.sh
sudo journalctl -u codekits-api -n 50 --no-pager

# OS updates:
sudo apt update
sudo apt upgrade -y
```

For Node patches, re-run scripts/install-node24.sh in planned maintenance, restart codekits-api and check both the local and HTTPS health routes.

If you need a file rollback, inspect the chosen snapshot with sudo tar -tzf /path/to/snapshot.tgz. Stop the API, then restore that trusted snapshot with sudo tar -C / -xzf /path/to/snapshot.tgz. Run daemon-reload, nginx -t, reload nginx and restart the API. This replaces the recorded paths; verify the snapshot first.

A file rollback does not roll back database changes. This demo has no update migrations. Real apps need a separate schema migration and recovery plan.

19 / Diagnose the failed layer

Work from the inside out: MySQL, the local API, Nginx, DNS, then HTTPS. A local API failure is not fixed by repeatedly requesting certificates.

SERVER - diagnostic commands

```
# API process and recent errors
sudo systemctl status codekits-api --no-pager
sudo journalctl -u codekits-api -n 100 --no-pager
curl -v http://127.0.0.1:3000/api/health

# Nginx configuration and errors
sudo nginx -t
sudo tail -n 60 /var/log/nginx/error.log

# MySQL and resource checks
sudo systemctl status mysql --no-pager
sudo journalctl -u mysql -n 60 --no-pager
free -h
df -h
sudo ss -lntp
```

502 Bad Gateway: Nginx cannot reach a usable upstream. Check codekits-api, its port and journal first.

503 with the sample JSON error: the API is reachable but a request failed; inspect its journal and MySQL.

Access denied: check DB_USER, DB_HOST and the matching MySQL account. Do not change the root authentication plugin to solve an app-account problem.

A React page with failed cards: inspect /api/notes in the browser. HTML instead of JSON often means the API location is missing or the wrong site block is active.

Certbot validation failure: confirm A and AAAA, public port 80, provider rules and the exact requested name. Build killed: check available RAM and disk.

20 / Quick command sheet

First-time DNS, Nginx installation and certificate issuance need the earlier sections. This page is a quick reference once you understand the path and context of each command.

SERVER - copyable reference

```
# Enter the source project
cd ~/codekits-tutorial/project

# First installation (after apt packages and firewall)
bash scripts/install-node24.sh
bash scripts/init-database.sh
bash scripts/deploy.sh

# Service operations
sudo systemctl status codekits-api --no-pager
sudo systemctl restart codekits-api
sudo journalctl -u codekits-api -f

# Nginx
sudo nginx -t
sudo systemctl reload nginx

# API
curl -fsS http://127.0.0.1:3000/api/health

# Certificate
sudo /snap/bin/certbot certificates
sudo /snap/bin/certbot renew --dry-run

# Backup
bash scripts/backup.sh "$HOME/backups"
```

Configuration: /etc/nginx/sites-available/codekits-demo; /etc/systemd/system/codekits-api.service; /etc/codekits-demo.env.

Public website files: /var/www/codekits-demo. API files: /srv/codekits-demo/api. Keep runtime secrets out of both the public build and source control.

21 / Sources, scope and validation

Prepared for CodeKits on 5 November 2026. The attached source and lockfiles are the reproducible starting point. The video teaches the process with on-screen instructions; it is not footage of an independently verified public deployment.

Validation performed for this package is recorded in VALIDATION.txt. Live Ubuntu provisioning, MySQL startup, provider networking, DNS propagation and certificate issuance still require a VPS and a real domain.

This example is intentionally a small read-only deployment. For a real account-based app, add authentication, authorization, input validation, abuse controls, monitoring and tested recovery suited to that app.

Official references:

Ubuntu Nginx installation: https://ubuntu.com/server/docs/how-to/web-services/install-nginx/

Ubuntu MySQL installation: https://ubuntu.com/server/docs/databases-mysql/

Ubuntu firewall: https://ubuntu.com/server/docs/how-to/security/firewalls/

Node.js release schedule: https://nodejs.org/en/about/previous-releases

Node.js downloads and verification: https://nodejs.org/en/download

Vite production build: https://vite.dev/guide/build.html

Express routing: https://expressjs.com/en/guide/routing/

MySQL2 connection pools: https://sidorares.github.io/node-mysql2/docs

MySQL database accounts: https://dev.mysql.com/doc/refman/8.0/en/create-user.html

systemd service configuration: https://manpages.ubuntu.com/manpages/noble/man5/systemd.service.5.html

Nginx proxy_pass: https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass

Nginx try_files: https://nginx.org/en/docs/http/ngx_http_core_module.html#try_files

Certbot for Nginx: https://certbot.eff.org/instructions?ws=nginx&os=snap