Ubuntu VPS Course

Flavio Copes

Updated August 3, 2026

Contents

Create and secure an Ubuntu VPS on DigitalOcean, connect it to a domain, serve HTTPS traffic, deploy an application, and keep it healthy.

What you'll learn: Provision a DigitalOcean Droplet, secure SSH access, configure Nginx and HTTPS, run an application with systemd, and operate the server safely.

Create the Droplet

Choose an Ubuntu image, region, size, SSH key, and recovery options without overspending.

What a VPS is

Understand the virtual machine you rent, the responsibilities the provider handles, and the administration work that still belongs to you.

A **Virtual Private Server**, or VPS, is a virtual machine running on provider hardware. DigitalOcean calls its virtual machines **Droplets**.

You get an operating system with its own users, processes, files, network interfaces, and public address. It behaves much like a physical Linux server, but DigitalOcean can create, resize, snapshot, or destroy it through a control panel or API.

Know the responsibility boundary

DigitalOcean operates the data center, physical hardware, and virtualization layer. You operate everything inside the guest system:

- user accounts and SSH access
- operating-system and application updates
- host firewall rules
- Nginx and application services
- application data and secrets
- backups, restore tests, monitoring, and incident response

This is the important mental model: a working Droplet is not a managed application. DigitalOcean can report that the virtual machine is powered on while your application is stopped, the disk is full, or the TLS certificate is broken.

Once we create the server, these commands will identify what we actually received:

```
cat /etc/os-release
uname -m
hostnamectl
```

The output should show the expected Ubuntu release, CPU architecture, and hostname. These facts belong in every useful support request.

A public IP address also creates a security boundary. Automated scanners will find open ports quickly. This does not mean the server is already compromised. It means updates, key-based access, a firewall, and useful logs are normal operating work, not optional hardening for later.

Your action is to write two lists. Put application code and generated build files under **rebuildable**. Put uploads, database data, secrets, and private keys under **must be protected**. That distinction will guide the deployment and backup decisions in the rest of the course.

Plan cost, region, and size

Choose the smallest useful server near its users and understand which resources keep billing until you remove them.

Choose the server from the workload you have now. A small site or learning project can normally start on a Basic shared-CPU plan. Dedicated CPU or a larger machine becomes useful when

measurements show sustained CPU pressure, memory pressure, or storage I/O problems.

Do not choose a size from traffic alone. A quiet application with a large in-memory process can need more RAM than a busy static site.

Make the placement decision

Choose a region close to the people and services that exchange the most data with the application. If the application uses a managed database, keep both in the same region unless you have a deliberate reason not to. This reduces latency and avoids sending private service traffic across distant networks.

Before creating anything, record:

- the region
- expected RAM, CPU, and disk needs
- whether backups and improved monitoring will be enabled
- the monthly estimate shown by the control panel
- any extra volumes, snapshots, or reserved addresses

Treat the control-panel estimate as the evidence. Prices and available plans change, so a copied price from an old tutorial is not reliable.

A powered-off Droplet still reserves resources and continues billing. Billing ends when you destroy the Droplet. Backups, snapshots, volumes, and reserved addresses are separate resources and can continue billing after the server disappears.

Starting too large wastes money. Starting too small without monitoring creates unexplained crashes. The safe approach is to begin with a reasonable minimum, enable metrics, and resize after you can point to the constrained resource.

Resizing upward is usually easier than moving to another region. A region change normally means creating another server, copying state, testing it, and moving traffic. Choose the region carefully now.

For your learner action, write a one-paragraph capacity guess for the notes application. Include the expected users, persistent data, region, initial plan category, and the metric that would justify a resize. We will compare that guess with real measurements later.

Create an SSH key

Generate a dedicated key pair, protect the private key, and add only the public key to DigitalOcean.

An SSH key pair has two parts. The private key proves your identity and stays on your computer. The public key can be installed on servers that should accept that identity.

Create a dedicated Ed25519 key on your computer:

```
ssh-keygen -t ed25519 -f ~/.ssh/digitalocean_notes -C "digitalocean-notes"
```

Choose a strong passphrase when prompted. The command creates:

```
~/.ssh/digitalocean_notes
~/.ssh/digitalocean_notes.pub
```

The file ending in `.pub` is the public key. Print and copy only that file into the DigitalOcean SSH-key form:

```
cat ~/.ssh/digitalocean_notes.pub
```

Before uploading it, inspect its fingerprint:

```
ssh-keygen -lf ~/.ssh/digitalocean_notes.pub
```

DigitalOcean should show the same fingerprint after saving the key. The fingerprint is shorter and safer to compare than a long public-key line.

Never print the private key. Do not paste it into DigitalOcean, a chat, the server, or a Git repository. If the private file is readable by other local users, OpenSSH may refuse it. Restore the expected permissions with:

```
chmod 600 ~/.ssh/digitalocean_notes
```

Be careful when choosing the filename. `ssh-keygen` asks before overwriting an existing key, but replacing a key can remove your only access to servers that trust it.

Back up the private key in an encrypted password manager or another protected backup. Also keep a second administrator key or a documented recovery-console path for an important server. Losing the only private key does not reveal it to an attacker, but it can lock you out.

Your action is to add the public key to DigitalOcean, compare the displayed fingerprint, and label it with the computer and purpose. A name such as `macbook-notes-admin-2026` is more useful later than `my key`.

Create the Ubuntu Droplet

Provision a current Ubuntu LTS image with the intended key, hostname, region, size, and recovery options.

Create a project in DigitalOcean, then create the Droplet inside it. A project keeps the server, volumes, and other related resources visible together.

Use the choices from the previous lesson:

- select the current Ubuntu LTS image
- choose the planned region and Basic plan
- select SSH-key authentication and the key you verified
- enable improved monitoring
- enable automated backups when the server will hold state you cannot recreate
- give the Droplet a role-based hostname such as `notes-web-1`

Do not select a prebuilt application image for this course. Starting from the Ubuntu image lets us see and control every installed service.

Verify what DigitalOcean created

After provisioning completes, record the public IPv4 address in your private operations notes. Confirm the control panel shows the intended region, image, size, project, key, backup setting, and monitoring setting.

Connect only after the Droplet reports that creation is complete. Once connected in the next lesson, run:

```
cat /etc/os-release
uname -m
hostnamectl --static
ip -brief address
```

The output should match the Ubuntu LTS release, architecture, hostname, and addresses you intended. Do not continue configuring a server when the basic identity is wrong.

A common mistake is selecting the wrong SSH key. Another is creating the server in a convenient-looking region that is far from the database or users. These are cheapest to fix now. If this brand-new Droplet contains no unique data, destroy it and create the correct one instead of building permanent work on the wrong foundation.

If the server contains data later, destruction is not a rollback. You would first create and verify a backup, build a replacement, test it, and move traffic.

For your learner action, save the Droplet IP and the expected SSH-key fingerprint in private notes. Then compare every control-panel choice with the capacity plan you wrote. Stop and correct any mismatch before the first login.

Check your understanding: creating a Droplet

Check VPS responsibility, DigitalOcean Droplets, cost, regions, sizing, SSH keys, Ubuntu LTS, hostnames, and backups.

Answer each question before continuing. Read the explanation after every answer, including the ones you get right.

Take this interactive quiz online

Secure access

Verify the host, create an administrator, protect SSH, and enable a firewall without locking yourself out.

Make the first SSH connection

Connect as root once, verify the host-key prompt, and distinguish the server identity from your user key.

The first connection establishes two identities. Your private key proves who you are. The server host key helps prove which machine answered.

Connect with the dedicated key and the Droplet IP address:

```
ssh -i ~/.ssh/digitalocean_notes root@203.0.113.10
```

Replace the example address with your Droplet address. OpenSSH displays a host-key fingerprint because this server is not in your local `known_hosts` file yet.

Do not accept an unfamiliar fingerprint automatically. Use the control-panel console available for the Droplet image and inspect the server's public host key through that independent path. If the Recovery Console requires a password, follow DigitalOcean's current root-password reset procedure first.

```
ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub
```

Compare the fingerprint character by character. Accept the connection only when it matches. OpenSSH then records the host key locally and should show an Ubuntu shell prompt.

Collect the first evidence:

```
whoami
hostnamectl --static
cat /etc/os-release
```

You should see `root`, the hostname you chose, and the expected Ubuntu release.

If authentication fails with `Permission denied (publickey)`, first check the username, IP address, and `-i` path. Then verify that the selected public key appears in the Droplet settings. Recreating an empty new Droplet with the correct key is often safer than weakening SSH authentication.

If OpenSSH later reports that the host identification changed, stop. A rebuild legitimately changes the key, but an unexpected change can mean the address now belongs to another server or the connection is being intercepted. Verify the new fingerprint through the recovery console before removing the old local entry.

Keep this root session open for the next lessons. Your action is to open a second terminal and make the same verified connection again. Two working sessions give you a recovery path while changing users, firewall rules, and SSH settings.

Create a sudo user

Stop using root for daily work by creating an administrator account and copying the authorized key with correct ownership.

Use root only to create the administrator account. Daily work should happen through a regular user that elevates one command at a time with `sudo`.

While logged in as root, create the account and add it to Ubuntu's `sudo` group:

```
adduser deploy
usermod -aG sudo deploy
id deploy
```

`id deploy` should list the `sudo` group. The password created by `adduser` protects local `sudo` use. SSH will continue to use the key.

Copy the working authorized key and give the new account ownership:

```
rsync --archive --chown=deploy:deploy ~/.ssh /home/deploy
chmod 700 /home/deploy/.ssh
chmod 600 /home/deploy/.ssh/authorized_keys
```

Inspect the result:

```
namei -l /home/deploy/.ssh/authorized_keys
```

Every directory in the path must allow `deploy` to reach the file, while `.ssh` and `authorized_keys` must not be writable by unrelated users.

Keep the root session open. From a second local terminal, connect as the new user:

```
ssh -i ~/.ssh/digitalocean_notes deploy@203.0.113.10
sudo whoami
```

The first command should open a shell as `deploy`. The second should print `root` after the password prompt. Also run `sudo -l` to see the privileges granted through the group.

Wrong ownership is the most common failure here. SSH may ignore a key file that another user can

modify. Repair ownership from the still-open root session:

```
chown -R deploy:deploy /home/deploy/.ssh
```

Do not close the original root connection until both key login and `sudo` work in a new session. If you do get locked out, use the DigitalOcean recovery console to repair the account rather than enabling weak remote authentication.

Your action is to complete the second-login test, then run `whoami`, `groups`, and `sudo whoami` and explain why all three outputs are different.

Enable UFW safely

Allow the active SSH path before enabling the firewall and verify the rule from a second connection.

UFW controls the host firewall inside Ubuntu. The safe order matters: allow the active SSH path, verify the rule, then enable filtering.

First inspect the packaged SSH application profile:

```
sudo ufw app info OpenSSH
```

It should describe the TCP port used by the SSH service. If you deliberately moved SSH to another port, allow that exact port instead of assuming the profile is correct.

Preview and apply the rule:

```
sudo ufw --dry-run allow OpenSSH
sudo ufw allow OpenSSH
sudo ufw enable
sudo ufw status verbose
```

The final output should report an active firewall and an allow rule for SSH. Keep the current SSH session open because an established connection can survive a mistake that blocks new ones.

From another terminal, make a new connection:

```
ssh -i ~/.ssh/digitalocean_notes deploy@203.0.113.10
```

The new login is the real verification. A rule displayed by UFW is not enough if it opens the wrong port or a DigitalOcean Cloud Firewall blocks the traffic earlier.

DigitalOcean Cloud Firewalls and UFW are separate layers. A connection must be allowed by both. Keep the provider firewall narrow too, but change one layer at a time so a failure has one likely cause.

If the second login fails, do not close the first session. Recheck `sudo ufw status numbered`, the SSH listening port with `sudo ss -lntp`, and any Cloud Firewall attached to the Droplet. Delete an incorrect UFW rule by its displayed number:

```
sudo ufw delete 2
```

If every SSH session is lost, use the recovery console and run `ufw disable` temporarily while repairing the intended rule. Do not leave the firewall disabled as the final fix.

Your action is to prove that a second SSH connection succeeds, then save the output of `sudo ufw status verbose` in your private operations notes.

Harden SSH without locking yourself out

Test key access first, change one server setting at a time, and keep a recovery path while reducing remote-login risk.

SSH hardening is safe only after the replacement access path works. Confirm again that `deploy` can log in with its key and run `sudo`. Keep that session open and keep the recovery console available.

Create a small drop-in instead of rewriting the package configuration:

```
sudoedit /etc/ssh/sshd_config.d/00-notes-hardening.conf
```

Add:

```
PermitRootLogin no
PasswordAuthentication no
KbdInteractiveAuthentication no
```

This removes direct root login and password-style remote login. It does not disable your working public-key login.

Validate the complete configuration before asking the service to read it:

```
sudo sshd -t
sudo sshd -T | grep -E 'permitrootlogin|passwordauthentication|kbdinteractiveauthentication|p
```

`sshd -t` should print nothing and exit successfully. The effective configuration should show root and password authentication disabled while public-key authentication remains enabled.

Only then reload SSH:

```
sudo systemctl reload ssh
sudo systemctl status ssh --no-pager
```

A reload preserves existing connections. Open a third terminal and verify a fresh `deploy` login. Also verify that root login is rejected:

```
ssh -i ~/.ssh/digitalocean_notes deploy@203.0.113.10
ssh -i ~/.ssh/digitalocean_notes root@203.0.113.10
```

The first should work and the second should fail.

A common failure is a typo or a setting in another drop-in that wins earlier. That is why both `sshd -t` and `sshd -T` matter: one checks syntax, while the other shows effective values.

If a fresh `deploy` login fails, keep the existing session open. Move the new file aside, validate again, and reload:

```
sudo mv /etc/ssh/sshd_config.d/00-notes-hardening.conf /root/
sudo sshd -t && sudo systemctl reload ssh
```

If every network session is gone, repair the file from the recovery console. Your action is to complete both positive and negative login tests before closing any earlier session.

Check your understanding: secure access

Check host keys, user keys, non-root administration, sudo, UFW order, cloud firewalls, SSH validation, and lockout recovery.

Answer each question before continuing. Read the explanation after every answer, including the

ones you get right.

Take this interactive quiz online

Serve the Web

Install Nginx, connect DNS, configure a site, and add HTTPS with a safe reload workflow.

Update Ubuntu and install Nginx

Apply the first updates, install the web server from Ubuntu repositories, and verify both the package and service state.

APT separates available-package information from installed packages. `apt update` refreshes the package index. `apt upgrade` reviews and installs newer versions for the current Ubuntu release.

Keep the working SSH session open and run:

```
sudo apt update
apt list --upgradable
sudo apt upgrade
sudo apt install nginx
```

Read the proposed upgrade before confirming it. A message about packages being kept back can be normal during Ubuntu phased updates. Do not work around it blindly by forcing individual versions.

Now verify the package, configuration, and service separately:

```
nginx -v
sudo nginx -t
systemctl is-enabled nginx
systemctl is-active nginx
sudo ss -lntp | grep -E ':80\s'
curl -I http://127.0.0.1
```

`nginx -t` should report successful syntax and file checks. The service should be enabled and active. `ss` should show a listener on port 80, and the local request should return an HTTP response such as `200 OK`.

This layered check matters. A running service does not prove the firewall permits traffic. A package being installed does not prove its configuration is valid.

If APT says another process holds its lock, check whether automatic updates are still running. Wait for the real package operation to finish. Do not delete lock files while APT or `dpkg` is active.

If Nginx fails to start, inspect evidence before reinstalling it:

```
sudo systemctl status nginx --no-pager
sudo journalctl -u nginx -b --no-pager
sudo nginx -t
```

A syntax error, missing file, or occupied port will usually appear there. Fix that cause and run `sudo systemctl restart nginx`.

Your action is to collect the successful `nginx -t`, `is-active`, listening-port, and local HTTP results.

The public request comes after we deliberately open the web ports.

Open HTTP and HTTPS

Expose only the web ports Nginx needs and keep SSH available while moving from the IP test to a real site.

Nginx installs UFW application profiles. Inspect the names and the ports behind the combined profile before using it:

```
sudo ufw app list
sudo ufw app info "Nginx Full"
```

`Nginx Full` should allow TCP ports 80 and 443. Port 80 is needed for ordinary HTTP and common certificate validation flows. Port 443 carries HTTPS.

Add the rule and inspect the complete public surface:

```
sudo ufw allow "Nginx Full"
sudo ufw status numbered
```

Keep the OpenSSH rule. The intended inbound list is now SSH, HTTP, and HTTPS. The application port we will use later must remain private on `127.0.0.1`.

Test HTTP from your own computer, not from the Droplet:

```
curl -I http://203.0.113.10
```

An HTTP response proves that the process, host firewall, provider firewall, and public route all agree. A local `curl` test could not prove those outer layers.

If the request times out while local HTTP works, compare `sudo ufw status`, the DigitalOcean Cloud Firewall, and the Droplet address. A provider firewall can block traffic before UFW sees it. If the connection is refused, check `sudo ss -lntp` and Nginx service state.

To roll back an accidental UFW rule, find its number with `sudo ufw status numbered` and delete that number. Never delete the SSH rule until a deliberate replacement access path has been tested.

Your action is to verify the three intended services in UFW and obtain an HTTP response from outside the server. HTTPS will not work yet; we first need a domain and certificate.

Point a domain to the server

Create the DNS records that map a hostname to the Droplet and verify resolution before requesting a certificate.

Choose the exact hostname that will serve the application, for example `notes.example.com`. At the authoritative DNS provider, create an `A` record whose value is the Droplet's public IPv4 address.

Add an `AAAA` record only when the Droplet has public IPv6 enabled, Nginx listens on IPv6, UFW permits it, and you have tested the complete IPv6 path. A broken AAAA record can make a healthy IPv4 service appear unreliable.

Verify authority and caches

Find the authoritative nameservers:

```
dig NS example.com +short
```

Ask one of those nameservers directly. For a zone hosted by DigitalOcean, one query can look like this:

```
dig @ns1.digitalocean.com A notes.example.com
```

Use a nameserver returned by your own `NS` query when another provider hosts the zone.

The answer should contain the Droplet address. Then compare a public recursive resolver and your normal local resolver:

```
dig @1.1.1.1 A notes.example.com +short
dig A notes.example.com +short
```

This tells you where a disagreement lives. If authority is wrong, fix the DNS zone. If authority is correct but a recursive resolver is old, wait for its cached answer to expire according to the previous TTL.

Do not keep changing the record while caches are converging. Each change creates another possible answer and makes diagnosis harder.

Once DNS resolves correctly, verify that Nginx receives the intended hostname:

```
curl -I http://notes.example.com
```

An HTTP response proves more than DNS alone. It checks resolution, the connection, firewall, and Nginx.

For a migration, keep the previous server working while old cached answers remain valid. The rollback is to restore the old record value and keep both endpoints available for at least the old TTL. DNS rollback is not instant for clients that already cached the new answer.

Your action is to record the authoritative answer, one public-resolver answer, the TTL, and the HTTP result. Do not request a certificate until all four point to the intended server.

Configure Nginx and HTTPS

Create a server block, test every configuration change, then obtain a trusted TLS certificate for the working domain.

Create a small document root so we can prove the hostname works before adding the application:

```
sudo install -d -m 755 /var/www/notes
echo '<h1>Notes server</h1>' | sudo tee /var/www/notes/index.html
sudoedit /etc/nginx/sites-available/notes
```

Add this server block, replacing the hostname:

```
server {
  listen 80;
  listen [::]:80;

  server_name notes.example.com;
  root /var/www/notes;
  index index.html;

  location / {
    try_files $uri $uri/ =404;
```

```
  }
}
```

Enable it, validate the complete configuration, and reload gracefully:

```
sudo ln -s /etc/nginx/sites-available/notes /etc/nginx/sites-enabled/notes
sudo nginx -t
sudo systemctl reload nginx
curl -I http://notes.example.com
```

`nginx -t` must succeed before the reload. The HTTP request should reach this server block. If you see the default Nginx page, check `server_name`, enabled-site links, and the hostname sent by the request.

Add HTTPS only after HTTP works

Install Certbot using the current official Nginx instructions linked in this lesson. Then request and install the certificate:

```
sudo certbot --nginx -d notes.example.com
```

Certificate issuance requires the hostname to resolve publicly to this server and the validation path to be reachable. A DNS mismatch, closed port 80, or proxy pointing elsewhere will make the request fail. Fix that layer instead of retrying repeatedly.

Verify the result from outside the Droplet:

```
curl -I https://notes.example.com
sudo certbot renew --dry-run
systemctl list-timers | grep -i certbot
```

The HTTPS request should present a trusted certificate for the hostname. The dry run proves the renewal path, not just today's certificate.

If an Nginx change fails, do not reload it. Repair the file or remove the new enabled-site link, run `nginx -t` again, and reload only a valid configuration. Keep a copy of the last working server block before later edits.

Your action is to save the successful configuration test, HTTPS response, certificate hostname and expiry, and renewal dry-run result in your operations notes.

Check your understanding: serving the Web

Check Ubuntu updates, Nginx service state, UFW profiles, DNS records, server blocks, configuration tests, HTTPS, and renewal.

Answer each question before continuing. Read the explanation after every answer, including the ones you get right.

Take this interactive quiz online

Run an application

Deploy code, protect environment values, create a systemd service, and reverse proxy requests.

Choose a deployment layout

Give application code and runtime files a predictable location and owner instead of running the project from a root home directory.

Separate the person who deploys from the process that runs the application. The `deploy` account can install releases through `sudo`. A dedicated `notes-app` account runs the service without an interactive login.

Create the service account and directories:

```
sudo adduser --system --group --home /var/lib/notes-app notes-app
sudo install -d -m 755 -o deploy -g notes-app /srv/notes-app
sudo install -d -m 755 -o deploy -g notes-app /srv/notes-app/releases
sudo install -d -m 750 -o notes-app -g notes-app /var/lib/notes-app
```

Use these responsibilities:

```
/srv/notes-app/releases/   versioned application code
/srv/notes-app/current     symlink to the active release
/var/lib/notes-app/        persistent application data
/etc/notes-app.env         production configuration and secrets
```

The service needs read and execute access to the active release. It needs write access only to explicit state directories such as `/var/lib/notes-app`. It should not own Nginx configuration, systemd units, or unrelated system files.

Check the boundary:

```
id notes-app
namei -l /srv/notes-app/releases
sudo -u notes-app test -w /etc && echo unexpected
sudo -u notes-app test -w /var/lib/notes-app && echo writable
```

The first write test should print nothing. The second should print `writable`.

A release directory makes rollback predictable. Deploy code to a new path such as `/srv/notes-app/releases/4f92c1a`, test it, then point `current` at it:

```
sudo -u deploy ln -sfn /srv/notes-app/releases/4f92c1a /srv/notes-app/current
readlink -f /srv/notes-app/current
```

If a release fails, point the symlink back to the previous directory and restart the service. This rolls back code. It does not undo a database migration, uploaded file change, or secret rotation, which need their own recovery plans.

A common mistake is letting the service write inside its release directory. That mixes disposable code with persistent state and makes rollback unreliable.

Your action is to create the account and directories, run both permission tests, and explain which paths belong in backups.

Install the runtime and application

Use a supported runtime source, deploy a pinned application revision, install production dependencies, and test it on localhost first.

Install the runtime version declared by the project. Follow the runtime's current official Ubuntu

instructions or a maintained package source whose update behavior you understand. Do not copy an old installation script into a root shell without reading where it comes from and what it changes.

Verify the tools before deploying:

```
node --version
npm --version
git --version
```

The Node version must satisfy the application's declared requirement. A successful command is evidence about the installed executable, not proof that the application supports it.

Create a release from a pinned Git commit:

```
cd /srv/notes-app/releases
git clone https://github.com/your-account/notes-app.git 4f92c1a
cd 4f92c1a
git checkout 4f92c1a
npm ci
git rev-parse HEAD
```

Replace the repository and commit with the real values. `npm ci` follows the lockfile and fails when it disagrees with `package.json`. Do not replace that useful failure with an unreviewed dependency update on the server.

If the server performs a production build, run the documented build and then use `npm prune --omit=dev` to remove build-only packages. If a trusted build system already produced the deployable artifact, install production dependencies with `npm ci --omit=dev` instead.

Then give the runtime account read access without making it the owner of the release:

```
sudo chown -R deploy:notes-app /srv/notes-app/releases/4f92c1a
sudo chmod -R g=rX,o= /srv/notes-app/releases/4f92c1a
```

Start it temporarily on localhost using the project's real start command:

```
sudo -u notes-app env HOST=127.0.0.1 PORT=3000 npm start
```

From another SSH session, verify the private endpoint:

```
curl --fail --show-error http://127.0.0.1:3000/health
sudo ss -lntp | grep ':3000'
```

The health request should succeed, and the listener must use `127.0.0.1`, not `0.0.0.0`. Stop the temporary process after the test. We will let systemd manage it next.

Common failures are a wrong runtime version, a missing production dependency, or a start command that assumes another working directory. Fix the release rather than opening the internal port publicly.

Your action is to record the commit hash, runtime version, clean install result, health response, and bind address. Keep the previous release directory for rollback.

Store environment values and secrets

Keep production configuration outside the repository and make secret files readable only by the service account.

Production configuration belongs outside the release directory. This lets the same code run in several environments and prevents a code rollback from restoring old credentials.

Create a root-managed environment file that the service group can read:

```
sudo install -m 640 -o root -g notes-app /dev/null /etc/notes-app.env
sudoedit /etc/notes-app.env
```

Use simple `NAME=value` entries:

```
NODE_ENV=production
PORT=3000
DATABASE_URL=replace-with-the-real-secret
```

Do not add `export`. A systemd `EnvironmentFile` is not an interactive shell script. When a value contains spaces or special characters, check systemd's environment-file quoting rules and test the application rather than guessing.

Verify ownership and mode without printing the contents:

```
sudo stat -c '%U %G %a %n' /etc/notes-app.env
sudo -u notes-app test -r /etc/notes-app.env && echo readable
sudo -u deploy test -r /etc/notes-app.env && echo readable-by-deploy
```

The first command should report `root notes-app 640`. The service should be able to read the file. Decide deliberately whether the deployment account belongs to the `notes-app` group and can read production secrets.

Never put the file in Git, a frontend bundle, process arguments, screenshots, or command output. Also avoid diagnostic commands that dump the complete service environment into logs or chat.

A common failure is making the file readable only by root while the service runs as `notes-app`. The service then starts without required values or fails with `Permission denied`. Fix the group and mode; do not run the application as root.

Secret rotation is a deployment. Update the provider credential first when possible, update the environment file, restart the service, verify health, then revoke the old credential. Keep a rollback window when the provider supports two active credentials.

Your action is to create the file with placeholder values, prove its permissions with `stat` and `test`, and write a separate `.env.example` containing names but no real values in the application repository.

Create a service and reverse proxy

Run the application with systemd and let Nginx handle public HTTP and HTTPS while forwarding to the private local port.

Systemd should own the application process. It starts the app after boot, records output in the journal, applies the service user, and restarts unexpected failures.

Create `/etc/systemd/system/notes-app.service`:

```
[Unit]
Description=Notes application
After=network.target

[Service]
```

```
Type=simple
User=notes-app
Group=notes-app
WorkingDirectory=/srv/notes-app/current
EnvironmentFile=/etc/notes-app.env
ExecStart=/usr/bin/node server.js
Restart=on-failure
RestartSec=5s
TimeoutStopSec=30s

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

Use the real absolute runtime path from `command -v node` and the application's real entry file. A service does not inherit your interactive shell setup.

Validate, load, and start it:

```
sudo systemd-analyze verify /etc/systemd/system/notes-app.service
sudo systemctl daemon-reload
sudo systemctl enable --now notes-app
sudo systemctl status notes-app --no-pager
sudo journalctl -u notes-app -n 50 --no-pager
curl --fail http://127.0.0.1:3000/health
```

The unit should be enabled and active. The local health request must succeed before Nginx is involved.

Now replace the static `location /` in the Nginx server block:

```
location / {
  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 $proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Proto $scheme;
}
```

Validate and reload:

```
sudo nginx -t
sudo systemctl reload nginx
curl -I https://notes.example.com
```

A `502 Bad Gateway` means Nginx could not get a usable response from the upstream. Check `systemctl status notes-app`, the journal, `ss -lntp`, and the configured port. Do not open port 3000 in UFW as a workaround.

For a code rollback, point `/srv/notes-app/current` at the previous tested release and restart `notes-app`. For a unit or Nginx rollback, restore the last working file, validate it, then reload or restart.

Your action is to reboot the Droplet during a safe learning window. After reconnecting, verify the

service is active and the public HTTPS health endpoint still works.

Check your understanding: running an application

Check deployment ownership, runtime versions, locked dependencies, localhost binding, environment files, systemd, and Nginx reverse proxying.

Answer each question before continuing. Read the explanation after every answer, including the ones you get right.

Take this interactive quiz online

Operate the server

Read logs, monitor resources, update, back up, recover, and retire the server responsibly.

Read the right logs

Follow a request through DNS, Nginx, the application service, and system logs without dumping unrelated data.

Debug one request through the layers instead of dumping every log on the server.

Start with a request that records timing and connection details:

```
curl --verbose https://notes.example.com/health
```

The output tells you whether DNS resolved, TLS completed, and HTTP returned a status. Note the exact time and status before moving to the server.

Nginx access and error logs normally live under `/var/log/nginx`. Application output managed by systemd appears in the journal. Reproduce the request while watching focused sources:

```
sudo tail -f /var/log/nginx/access.log /var/log/nginx/error.log
sudo journalctl -u notes-app --since "10 minutes ago" -f
```

Use separate terminals so the events remain readable. A request in the access log proves it reached Nginx. An Nginx error such as `connect() failed` points toward the local upstream. An application journal entry proves the request reached the service.

If nothing appears, work outward:

```
dig +short notes.example.com
sudo ufw status
systemctl is-active nginx notes-app
sudo ss -lntp | grep -E ':80|:443|:3000'
```

This separates DNS, firewall, service, and listening-port failures.

A common mistake is reading old messages from a previous failure. Filter by the reproduction time and current boot:

```
sudo journalctl -u notes-app -b --since "2026-07-30 12:00"
```

Replace the time with the real event. Another mistake is restarting before collecting evidence. A restart can remove the failing state and make the cause harder to find.

Logs are sensitive data. Do not record passwords, authorization headers, session cookies, complete

request bodies, or environment variables. Redact secrets before sharing a useful excerpt.

Your action is to request a missing path such as `/does-not-exist` and trace the resulting status through curl, the Nginx access log, and the application journal. Write down the last layer that observed it.

Monitor capacity and availability

Watch external reachability, disk, memory, load, and service health before a small problem becomes downtime.

Monitoring answers two different questions: can users reach the application, and does the server have enough capacity to keep serving it?

Monitor the public HTTPS endpoint from outside the Droplet. The check should require a successful status and run often enough to notice useful downtime:

```
curl --fail --silent --show-error https://notes.example.com/health
```

An internal process check cannot detect broken DNS, a provider firewall, an expired certificate, or a failed route. Keep an external check even when DigitalOcean reports that the Droplet is powered on.

Enable DigitalOcean improved metrics and create alert policies for the resources relevant to this server. On the Droplet, collect a baseline during normal use:

```
df -h
free -h
uptime
ps -eo pid,comm,%cpu,%mem --sort=-%mem | head
systemctl is-active nginx notes-app
```

`df` shows filesystem capacity. `free` shows memory and swap. `uptime` includes load averages. Process data helps connect pressure to an owner.

Do not alert on one universal number copied from a tutorial. Choose thresholds that leave time to respond. A disk alert at 99% is too late when logs can grow quickly. A short CPU spike may be harmless, while sustained high CPU with slow requests needs investigation.

When an alert fires, verify the user-visible symptom and the resource trend before resizing. A full log directory needs cleanup and retention. A memory leak needs a code or process fix. A slow external API will not become healthy because the Droplet has more RAM.

Write a small response for each alert: where to look, what can be cleaned or restarted safely, when to resize, and who receives the notification. Monitoring without an action becomes noise.

Your action is to save today's disk, memory, load, and process baseline, then configure one external availability alert and one capacity alert. Trigger or test the notification path so you know it reaches you.

Updates, backups, and restore

Patch the system, protect persistent data, and practice recovery instead of treating a provider snapshot as the entire backup strategy.

Updates reduce known risk. Backups reduce the damage when an update, deployment, operator,

disk, or provider event destroys useful state. Neither one replaces the other.

Patch deliberately

Ubuntu Server normally applies security updates through `unattended-upgrades`, but you still need to inspect its result and plan service restarts:

```
sudo apt update
apt list --upgradable
sudo apt upgrade
test -f /run/reboot-required && cat /run/reboot-required.pkgs
sudo systemctl --failed
sudo tail -n 50 /var/log/unattended-upgrades/unattended-upgrades.log
```

Run updates during a window when you can verify the application. Keep a working SSH session open. Afterward, test the public health endpoint and the main user workflow. If a reboot is required, announce it, verify backups first, reboot, and repeat the checks.

Back up state, not just the virtual machine

Inventory everything needed to recover:

- application uploads and persistent files
- database data using a database-aware backup
- `/etc/notes-app.env`, systemd units, and Nginx configuration
- the deployed commit and build instructions
- DNS, firewall, monitoring, and external-service settings

DigitalOcean backups and snapshots are useful recovery layers. They are not the only copy for critical data. Keep another protected copy in a separate failure domain with deliberate retention and encryption.

Check that backup files exist, are recent, and have non-zero size, but do not stop there. Create a disposable Droplet or isolated restore location. Restore the application data and database, supply test credentials, start the services, and run integrity and application checks.

The strongest evidence is a successful restore, not a green backup job. Common failures include omitting database consistency, keeping the only backup beside the source server, or discovering that the restore instructions require a secret nobody retained.

If an update breaks the application, first restore the last working release and configuration. Use a full-server restore only when the system state itself cannot be repaired safely. Database rollback needs a tested database recovery plan; code rollback does not undo schema changes automatically.

Your action is to choose one disposable file and one database record, back them up, restore them somewhere isolated, and record the commands and elapsed time. That becomes the first version of the recovery runbook.

Recover or retire the Droplet

Use the recovery console carefully, rebuild compromised systems, and remove every billable resource when the server is no longer needed.

Use the DigitalOcean recovery console when the Droplet is running but SSH is unavailable. It gives you an independent path for repairing access without opening insecure network authentication.

Start by collecting evidence:

```
df -h
sudo systemctl --failed
sudo systemctl status ssh --no-pager
sudo sshd -t
sudo ufw status numbered
sudo journalctl -u ssh -b --no-pager
```

A full disk, invalid SSH configuration, wrong firewall rule, or failed service needs a different fix. Change only the failing layer. If a new SSH drop-in caused the outage, move it aside, validate, reload SSH, and test a fresh remote connection before leaving the console.

Do not treat suspected compromise as an ordinary configuration failure. Preserve the relevant logs and timeline, isolate the server when possible, and rotate credentials from a trusted computer. Rebuild from a known image and verified data backup. Removing one suspicious process does not prove an attacker left no other access.

Retire a server without leaving loose ends

Before destruction:

1. move DNS or traffic to the tested replacement
2. keep the old service available until relevant DNS caches expire
3. make and restore-test the final data backup
4. copy only the logs required by your retention policy
5. rotate credentials that will no longer be used
6. verify no jobs or integrations still call the old address

Then destroy the Droplet in the control panel. Powering it off does not end billing.

Inspect the project afterward for resources that have their own lifecycle: volumes, snapshots, backups, reserved IPs, load balancers, and firewalls. Remove only the resources you have positively identified as unused. A snapshot kept for retention may be intentional; an unattached volume can still contain both sensitive data and ongoing cost.

The destructive action has no simple rollback. Confirm the Droplet name, project, address, replacement health, and final backup before clicking Destroy.

Your action is to write two short runbooks before you need them: “SSH unavailable” and “retire this Droplet.” Include the recovery-console path, verification commands, backup location, DNS owner, and exact evidence required before destruction.

Check your understanding: operating the server

Check logs, external monitoring, capacity, updates, backups, restore tests, recovery, compromise response, destruction, and billing.

Answer each question before continuing. Read the explanation after every answer, including the ones you get right.

Take this interactive quiz online

The interactive version of this course is available at https://flaviocopes.com/courses/vps/.