[redacted]OS architecture setup

Hetzner Virtual Private Server (Dedicated)

Paddy's note: I've used Hetzner before, the internal ops platform I showed Jay and Jack before I joined was on a shared hetzner VPS. By using a dedicated VPS we make sure that other people's application don't eat into resources we need to run the system, and we protect the application from Directory Traversal, a technique whereby a hacker can hack an unsecured website or web application on the same server, and use that to access a different one. Shared hosting would undermine any security we build into the application.

https://www.hetzner.com/cloud

CCX23AMD
4 vCPUs
16 GB RAM
160 GB SSD
20 TB Traddic
€ 0.0392
€ 21.40 + usage per month

Addition: Server security is pretty good with Hetzner, is a built in firewall so we can limit access to the server via IP address.

Cloudflare R2 Storage

Paddy's note: One of the cheaper options, Cloudflare only charges for storage, whereas other solution providers charge for "egress" downloading and reading stored data. We would only need to use this minimally and it integrates with laravel nicely. This would be a good time to move the [redacted]main onto cloudflare as we will want the DDOS and Bot protection applied to the [redacted] system.

https://developers.cloudflare.com/r2/pricing/

R2 pricing

	Standard storage	Infrequent Access storage
Storage	$0.015 / GB-month	$0.01 / GB-month
Class A Operations	$4.50 / million requests	$9.00 / million requests
Class B Operations	$0.36 / million requests	$0.90 / million requests
Data Retrieval (processing)	None	$0.01 / GB
Egress (data transfer to Internet)	Free [1]	Free [1]

There is also a free tier, I would suggest using that to start and only upgrading if we have to.

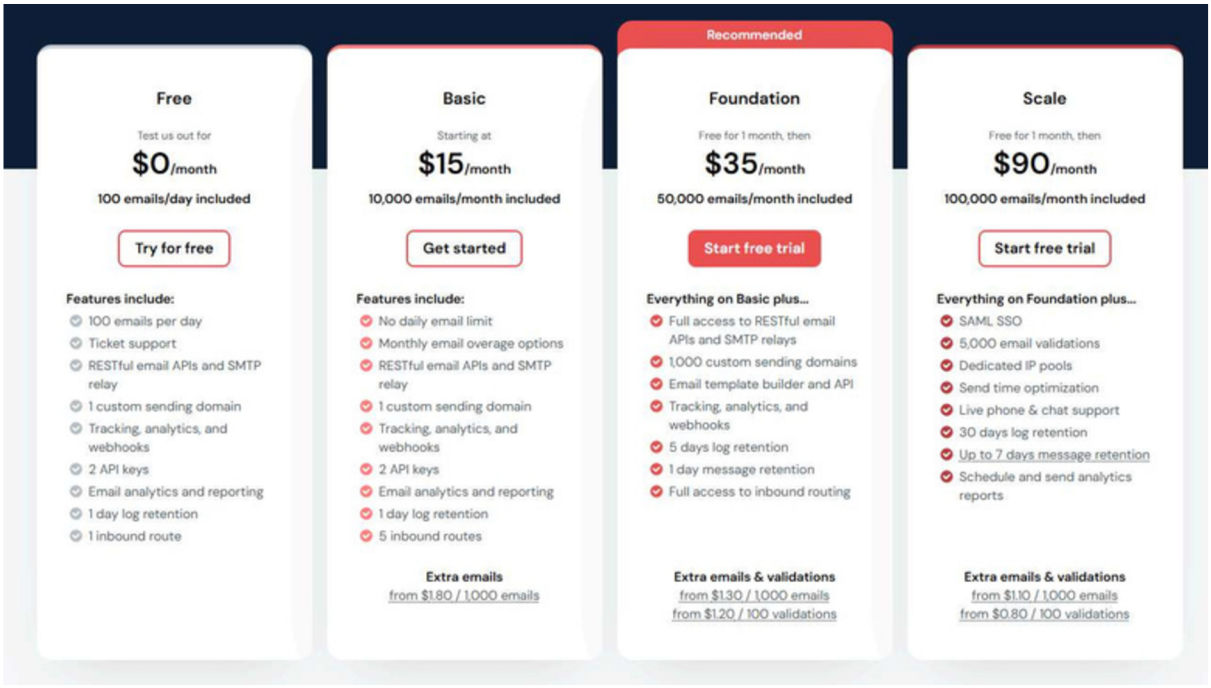
Free tier

You can use the following amount of storage and operations each month for free.

	Free
Storage	10 GB-month / month
Class A Operations	1 million requests / month
Class B Operations	10 million requests / month
Egress (data transfer to Internet)	Free [1]

Mailgun

Paddy's note: Using a dedicated service like Mailgun is an essential security and reliability layer. Direct-from-server emails are increasingly blocked by enterprise firewalls to prevent phishing. By using Mailgun, we offload the liability of email delivery to a specialist, ensuring that 100% of internal operations alerts are received instantly while simultaneously preventing any unauthorized 'spoof' emails from being sent on behalf of the company. Mailguin has a free tier, I suggest we go with that and see how we go.



Free	Basic	Foundation (Recommended)	Scale
Test us out for	Starting at	Free for 1 month, then	Free for 1 month, then
$0/month	$15/month	$35/month	$90/month
100 emails/day included	10,000 emails/month included	50,000 emails/month included	100,000 emails/month included
Try for free	Get started	Start free trial	Start free trial
Features include:	Features include:	Everything on Basic plus...	Everything on Foundation plus...
100 emails per day	No daily email limit	Full access to RESTful email APIs and SMTP relays	SAML SSO
Ticket support	Monthly email overage options	1,000 custom sending domains	5,000 email validations
RESTful email APIs and SMTP relay	RESTful email APIs and SMTP relay	Email template builder and API	Dedicated IP pools
1 custom sending domain	1 custom sending domain	Tracking, analytics, and webhooks	Send time optimization
Tracking, analytics, and webhooks	Tracking, analytics, and webhooks	5 days log retention	Live phone & chat support
2 API keys	2 API keys	1 day message retention	30 days log retention
Email analytics and reporting	Email analytics and reporting	Full access to inbound routing	Up to 7 days message retention
1 day log retention	1 day log retention	Extra emails & validations	Schedule and send analytics reports
1 inbound route	5 inbound routes	from $1.30 / 1,000 emails	Extra emails & validations
Extra emails	Extra emails	from $1.20 / 100 validations	from $1.10 / 1,000 emails
from $1.80 / 1,000 emails	from $1.80 / 1,000 emails		from $0.80 / 100 validations

Additional Notes

- Server-side firewalling is handled with Hetzner's built-in firewall. Cloudflare will take care of the application firewall. These are crucial. And both will need ongoing monitoring.
- Multifactor authentication should be implemented by default. Ideally with an authenticator app or Yubi key. Lararavel's own fortify package facilitates this.
- Principles of Least Access should apply to data at all levels, s[redacted]HQ and Site admins (us) can see everything and go everywhere, owners/managers of specific gyms can only see data pertaining to their gym, general gym staffers should only be able to see data relevant to their role. To facilitate we will be using Spatie Laravel Permissions, this is a solid, well maintained and well developed developed Laravel package that may as well be part of Laravel's framework anyway.
- Fail2ban should be installed and Laravel's built-in rate-limiter will protect against brute force attacks on the server, Laravel's rate limiter also works with cloudflare to prevent attacks on the application.
- All data rendered on the website must be rendered from the database where practical, this means that data can be encrypted at rest which is a legal requirement. API data for now doesn't need to be saved to the database, as currently that is VirtuaGym's problem.
- We will set appropriate security headers both on the server and in the application itself.
- Aim to transition away from storing API keys, secrets and similar in plaintext in the environment varials and towards Cloudflare secret store or similar. This allows up to 20 secrets for free and prevents these keys being compromised if the server is breached. Cloudflare Secret Store is currently in open Beta, but as we need cloudflare for other things, it makes sense to keep as much as possible in that family of services.
- Ideally we will push our commits to github, and when those changes are merged with the main branch we can use github actions to push that code onto the server. Within these github actions we can implement whatever tests are needed, my recommendation is to implement Automated Dependency Auditing so that if a known CVE is detected in the codebase during commit, further syncing with the server is frozen until the issue is resolved. Some manual monitoring is needed as some of these will be false flags as other precautions taken elsewhere will inevitably render some of these as non-issues.
- We will need to create an immutable audit log for continuous security monitoring. This can be stored in the client's object storage, we can still access this, but as it is ultimately client data, it will need to remain isolated on their infra.
- Laravel handles input validation and sanitization natively, as well as many other attack vectors. Coding patterns should conform to Laravel best practices where possible.