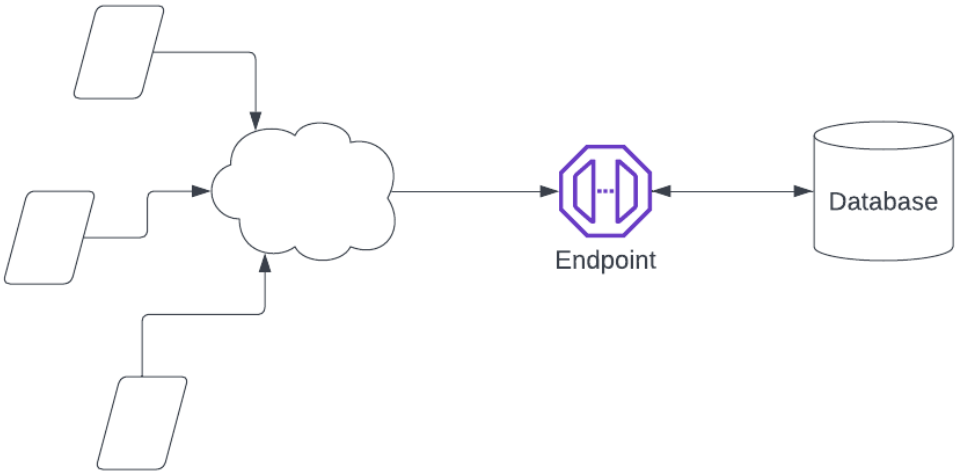


An infrastructure of 1x Heroku dyno with a hobby tier PostgreSQL database is not tenable for a forecasted load of 20,000 clients posting to a singular API instance. Assuming a datastore that realistically would encounter storage limits (10,000 rows) after half of the fleet has sent a single datapoint, this deployment would not scale before other potential bottlenecks arise.

At a minimum, vertical scaling of the existing database would need to be implemented to accommodate the additional data footprint that a client base of 20,000 devices would introduce. Row limitations are eliminated on the standard and premium plans. To determine how much we should scale vertically, it is important to understand to what degree we would need to fan out.

Given the sampling rate of 1req / 500ms, we would want to keep the p95 time to a reasonable level, so as to not backlog a considerable volume of requests to our service. This can be accomplished in a few ways.

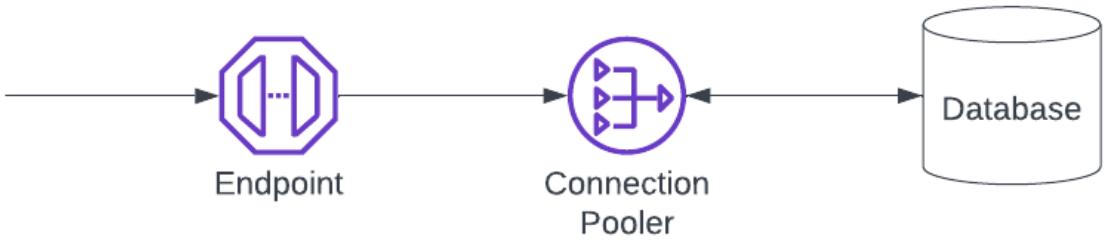
To enable horizontal scaling, our application would need to at least scale beyond the hobby tier dynos to those in the standard offering, as a singularly replicated dyno would not be able to accommodate the fleet of devices that we wish to service. However, if we use the standard tier offering – this scaling needs to be done in a more manual fashion. Nodes need to be added and recycled as necessary which may not be desirable.

One strategy which may help, would be to enable autoscaling with a threshold of 500ms response time, to address the increased load throughout the morning (peak coffee consumption), and gradually cool off as the service receives less traffic in the afternoon. Autoscaling is only possible on the premium tier. I would propose the lowest such offering (Performance M dyno) which comes with 2.5GB of RAM and support for unlimited process types. Assuming a conservative estimate of 300-400MB of memory utilization for a Ruby on Rails application, we could realistically accommodate 3-5 worker tenancy on a dyno (assuming a multi-core processor on the node) we could expect to be able to concurrently handle 12-20 requests at a given time on a single node.

With most Heroku Postgres offerings containing a ceiling of 400-500 concurrent connections at any given time, the maximum fleet size could realistically scale to 20-25 instances (though, this likely would

not occur in practice if we assume an even distribution of brewing devices across the United States and multiple time zones where only a sample of the deployed fleet would be under heavy load at any given time.)

Introducing a middleware pooling layer between the application and the database (as shown below) with pgBouncer may also be helpful to accommodate period of heavy database utilization without stalling the web servers significantly. Heroku's offering allows a maximum of 10,000 such connections to be pooled, i.e. half of the deployed fleet of coffee brewing devices.



Proposal: From the existing software architecture (hobby dynos and hobby PostgreSQL) I believe we need to scale both horizontally and vertically to accommodate the additional load from the fleet of coffee devices. First, we should upgrade our dynos to Performance M tier to take full advantage of autoscaling based on p95 response time and using our expected sampling rate (1req / 0.5s) as a threshold for this configuration. Our database should also be scaled from the hobby tier to at least the standard-3 tier plan to both remove the existing row limit of our database and introduce a reasonable connection limit. As an additional measure, we should add a pgBouncer connection pooling middleware layer to enable our application to fan out towards a sensible threshold while ensuring the database traffic is maintained within the bounds of its compute resources.