Key Metrics

Here is my suggestion for an actionable and measurable set of goals, based on the priorities you have explained to me.

1. **Overprovisioning**: Reduce vCPU and memory allocated in Fargate by 25% (175 vCPU, 500 GiB, $6500 USD per month in vCPU, $2000 USD per month in memory) to 500 vCPU and 1.5 TB.
2. **Quality of Service**: 99.9% SLO uptime on web request queue time of p95 10ms (100ms for core and any other customer-facing services), 99.9% SLO uptime of 4 SLO based queues (within_0_seconds, within_5_minutes, within_1_hour, within_24_hours) in Sidekiq for all services.
3. **Uptime**: 99.99% uptime based (ping) SLO for web services.

I have the following questions remaining to resolve on key metrics:

1. What is the right number of 9s for 2 and 3?
2. Can we get accurate metrics for CPU and memory allocated by Fargate in Datadog so we can commit and track caps? Sidekiq total thread count?
3. How will we roll out the infrastructure changes needed for #2 to the 2-dozen-or-so apps that will start meeting these standards? What have you already done here?
4. How do you want to establish a work cadence? Who will perform the work and how will it be tracked?

Overprovisioning

Here are the details of how to accomplish the Overprovisioning goal.

Deploy and tune application auto scaling

- We need to capture and publish request queue time to CloudWatch so that a scaling policy can use it. What is published today?
- Core and other customer facing services (i.e., services which are NOT called by other services) can have a 100ms p95 target for request queue time, but any internal service this needs to be lower (10ms suggested)
- End goal: all services (web and sidekiq) autoscale using queue times.

Change task sizing

- Core needs 12-75 Puma processes during the average day, so tasks of (WEB_CONCURRENCY) 12 would work well
- Core sidekiq should be 1-2 processes, but it currently is not due to bad sleep code in CreateWebhookProcessesWorker. It is currently > 30 processes
- You have many services, but most should be 1 pod each 99% of the time on web/sidekiq. There will inevitably a lot of waste caused by this (versus a monolith).

    - Web: 1x4proc, so probably 4 cpu/8gb pods
    - Worker: 2 task types, 1 process each. One task type for each of the low/high SLOs, 0.5 CPU each request I should think. High SLOs can scale to zero
- End goal: All services use a "standard" task size and format, except for Core. All services use SLO queues.

Quality of Service

These are the details about how to accomplish the Quality of Service goal.

SLOs and monitors

- p95 100ms request queue time for core, 10ms for any subservice
- Deployed via Terraform
- A request queue time monitor covers both full outage and partial (brownout) outage conditions
- End goal: each service has Datadog SLOs and monitors for queue time (web/sidekiq)

Migrate to SLO queues

- On most services, these will still be served by two process types, maybe assign half a CPU to each (1 hour SLO)
- Use four queues, based on SLO of time spent in the queue:
    - within_0_seconds
    - within_5_minutes
    - within_1_hour
    - within_24_hours
- End goal: each service uses SLO queues for Sidekiq

Puma and Sidekiq threadpool tuning

- Let's install gvl metrics middleware to get an idea of what the correct thread count will be for various services. Speedshop will make recommendations for thread count tuning based on this (and you will hopefully get access to our auto-tuning beta soon)
- It's possible that the "high SLO" task type should be 1 thread by default, purely to manage concurrency and err on the side of safety rather than throughput.
- End goal: Each service is running 3 threads on web and 5 on Sidekiq "low SLO"/1 on "High SLO" (1hr+) OR to a number "tuned" by speedshop.

Puma and Sidekiq CPU/memory container sizing

- We should deploy jemalloc to all services, this alone will probably cut memory allocation in half.
- End goal: All services have jemalloc (can we enforce via a Datadog Service Check?)

Needs a standardized gem/approach for all services

- Request queue time/sidekiq queue time -> Application Auto Scaling
- SLO queues for Sidekiq
- jemalloc in Dockerfile
- Terraform: SLOs and monitors (Uptime and Quality of Service)
- End goal: There is a single place where we can check compliance of all these points for every service, and a set of tools (or one tool) that is used by all services.

Uptime

This goal is somewhat overlapping with the previous, but it does have some specific tasks.

Set up SLOs and monitors

- Based on Synthetics tests/pings
- Set up in Terraform
- End goal: Every service has a monitor and SLO

Process improvements

- Add me to future postmortems. The causes of these outages will be quite wide and difficult to predict, so probably my best involvement here will just be to get in on your postmortems.

Worker optimization

- Reduce Sidekiq worker concurrency to minimum levels necessary
  - SLO queues will help with this.
- End goal: Total Sidekiq thread count across the infrastructure reduced by 50%. I'll create a Datadog chart/notebook for tracking (don't have the exact number)