# Work within the limits


- Choose a compute size
- Understand account quotas

- Count existing sandboxes {#1-count-existing-sandboxes}
- Back off after a refusal {#2-back-off-after-a-refusal}


Bound work so account limits and temporary capacity refusals do not turn into unbounded retries.

Use an authenticated client. Account limits are separate from SDK configuration; changing the client does not grant additional capacity.

## Choose a compute size

Cloud Sandboxes uses fixed CPU and memory pairs as billing shapes:

| Size | vCPUs | Memory | SDK memory value (MiB) |
|---|---|---|---|
| Micro | 1 | 2 GiB | 2048 |
| Small | 2 | 4 GiB | 4096 |
| Medium | 4 | 8 GiB | 8192 |
| Large | 8 | 16 GiB | 16384 |
| XL | 16 | 32 GiB | 32768 |

Choose a size by name, for example, `resources: 'small'`. The supported names are `micro`, `small`, `medium`, `large`, and `xl`. Kit launches default to Small when you omit resources. Explicit settings take precedence.

You can still supply both CPU and memory directly. Use a supported pair; 1 CPU with 1024 MiB is not supported in Cloud Sandboxes. A saved image supplies its own resources, so do not override them when creating from that image. Your account and available capacity determine whether a request can be accepted. Check your Docker billing terms for prices.

## Understand account quotas

The default limits are 10 concurrent sandboxes, 50 stored sandboxes, 100 volumes, 100 secrets, and 3 images being prepared at once. These are account-wide defaults, not a separate allowance per user. Your account can have different limits; confirm them with Docker before sizing a large workload.

Stopping an ordinary sandbox frees its concurrency slot but leaves it in stored usage. Starting or resuming it needs a slot again. An always-on sandbox, configured to restart automatically on timeout, retains its concurrency reservation even while stopped. Delete sandboxes you no longer need to release stored usage.

# Count existing sandboxes {#1-count-existing-sandboxes}

Walk all sandbox pages and count resources visible to you. This gives your application a usage observation, not a reservation or authoritative quota balance. Other users can hold resources in the same account that your list does not show.

Another caller can create a sandbox immediately afterward. Treat the create response as the final decision, even when your count appears below a planned limit.

{{< tabs >}} {{< tab name="TypeScript" >}}

```
let count = 0;
for await (const sandbox of client.all({ pageSize })) {
  if (sandbox.name) count++;
}
return count;
```

**Complete TypeScript example: limits/budget.ts**

```
import type { Sandboxes } from '@docker/sandboxes';

export async function countSandboxes(client: Sandboxes, pageSize: number) {
  let count = 0;
  for await (const sandbox of client.all({ pageSize })) {
    if (sandbox.name) count++;
  }
  return count;
```

```
}
```

{{< /tab >}} {{< /tabs >}}

# Back off after a refusal {#2-back-off-after-a-refusal}

Use the server's retry delay when supplied and keep the same idempotency key for the same create request. Bound the number of attempts and give the overall operation a deadline.

The example owns the retry loop, disables automatic SDK retries, and sets an overall timeout. Do not layer two retry policies without accounting for their combined attempts and deadlines.

A quota refusal may need cleanup or an account change rather than another immediate request. Stop after the bound, report the failure, and keep any accepted sandbox identity available for inspection.

{{< tabs >}} {{< tab name="TypeScript" >}}

```
for (let attempt = 0; attempt < attempts; attempt++) {
  try {
    return await client.kits.launch(
      'shell',
      { displayName, resources: { cpus: 2, memoryMib: 4096 } },
      {
        idempotencyKey: requestId,
        timeoutMs: 300_000,
        signal,
        maxRetries: 0,
      },
    );
  } catch (error) {
    if (
      !(error instanceof RequestError) ||
      error.raw.code !== 'resourceExhausted' ||
      attempt + 1 === attempts
    )
      throw error;
```

```
    await pause(retryAfter(error, fallbackMs), undefined, { signal });
  }
}
throw new Error('attempt count was validated');
```

### Complete TypeScript example: limits/backoff.ts

```
import { setTimeout as pause } from 'node:timers/promises';
import {
  RequestError,
  type Sandbox,
  type Sandboxes,
} from '@docker/sandboxes';

export async function createWithBackoff(
  client: Sandboxes,
  displayName: string,
  attempts: number,
  fallbackMs: number,
  requestId: string,
): Promise<Sandbox> {
  if (!Number.isInteger(attempts) || attempts < 1)
    throw new Error(`attempts must be at least 1, got ${attempts}`);
  const signal = AbortSignal.timeout(300000);
  for (let attempt = 0; attempt < attempts; attempt++) {
    try {
      return await client.kits.launch(
        'shell',
        { displayName, resources: { cpus: 2, memoryMib: 4096 } },
        {
          idempotencyKey: requestId,
          timeoutMs: 300_000,
          signal,
          maxRetries: 0,
        },
      );
    } catch (error) {
      if (
        !(error instanceof RequestError) ||
```

```
        error.raw.code !== 'resourceExhausted' ||
        attempt + 1 === attempts
      )
        throw error;
      await pause(retryAfter(error, fallbackMs), undefined, { signal });
    }
  }
  throw new Error('attempt count was validated');
}

export function retryAfter(
  error: RequestError,
  fallbackMs: number,
): number {
  const header = error.retryAfter?.trim();
  if (header) {
    const delay = /^\d+(?:\.\d+)?$/.test(header)
      ? Number(header) * 1000
      : /^(?:Mon|Tue|Wed|Thu|Fri|Sat|Sun)[a-z]*,?\s/.test(header)
        ? Math.max(0, Date.parse(header) - Date.now())
        : NaN;
    if (Number.isFinite(delay) && delay >= 0) return delay;
  }
  return error.decoded.retryDelayMs ?? fallbackMs;
}
```

{{< /tab >}} {{< /tabs >}}

---

□□□ #2
□ □□□ □□□ 2026-10-31 18:57:21 UTC
□ □□□ □□□ 2026-10-31 18:57:23 UTC