Reliable Software Systems

Week 7: Scalable API Design

February 21, 2018
Alyssa Pittman
University of Washington Allen School

Vote for week 10 topic!

The topic for the last week of class will be “Student’s Choice” -

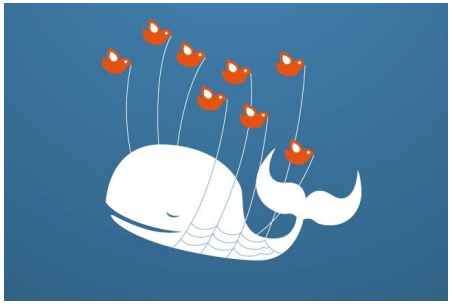
Vote here! https://bit.ly/2GUSyU8

Voting will be open through Friday in case you need more time, then I’ll send out the results.

Motivating Example: Twitter v1 API

Twitter launched in 2006 and was well-known for a few reasons, including:

- The fail whale
- An easy-to-use API



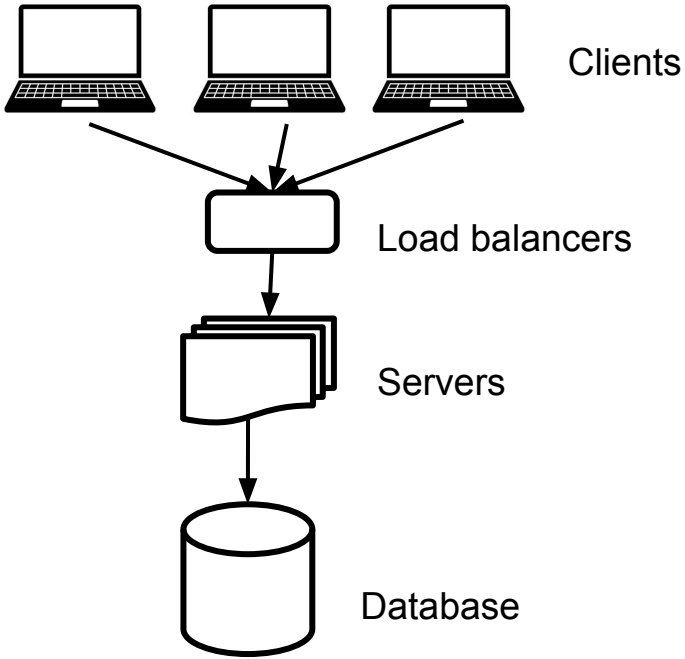
Increased API usage was responsible for some outages

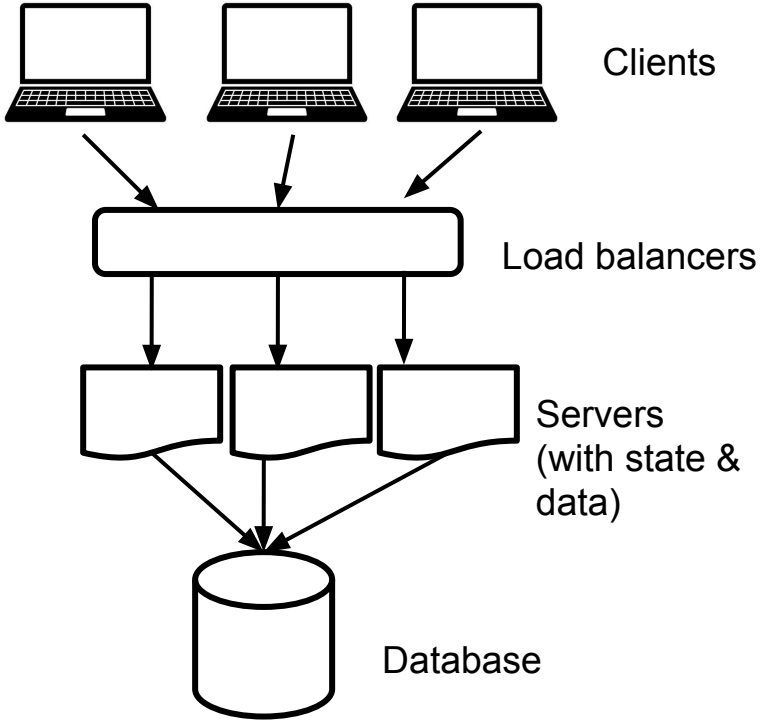
In 2013, Twitter retired its v1 API, replacing it with an API required OAuth

Developers still haven't forgiven Twitter for making them move

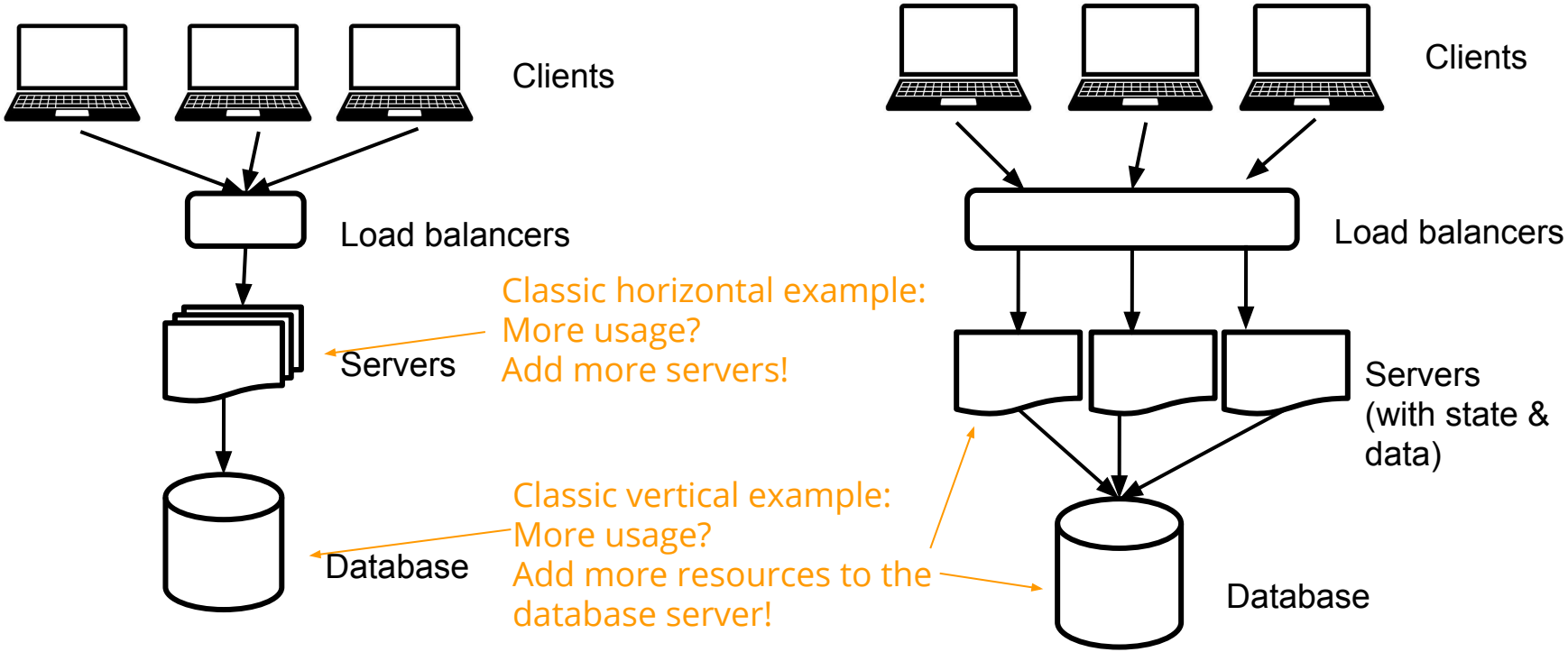
https://web.archive.org/web/20130513094200/https://dev.twitter.com/blog/application-only-authentication

Stateless vs stateful services





Horizontal vs vertical scaling



Example API definitions

REST: usually object-oriented

GitHub commit comments API:

Create
`POST /repos/:owner/:repo/commits/:sha/comments`

Get
`GET /repos/:owner/:repo/comments/:comment_id`

Update
`PATCH /repos/:owner/:repo/comments/:comment_id`

Delete
`DELETE /repos/:owner/:repo/comments/:comment_id`

List
`GET /repos/:owner/:repo/comments`

https://developer.github.com/v3/repos/comments/

RPC: often action-oriented

channels

Get info on your team's Slack channels, create or archive channels, invite users, set the topic and purpose, and mark a channel as read.

Method	Description
channels.archive	Archives a channel.
channels.create	Creates a channel.
channels.history	Fetches history of messages and events from a channel.
channels.info	Gets information about a channel.
channels.invite	Invites a user to a channel.
channels.join	Joins a channel, creating it if needed.
channels.kick	Removes a user from a channel.
channels.leave	Leaves a channel.

https://api.slack.com/methods#channels

Limit data: pagination

Paginate list responses rather than returning all objects.

Possible implementations:

- Explicit pages (example: get page 2)
- User-specified pages (example: get page 2, pagesize 100)
- Cursor (example: get after xyz, pagesize 10)

Limit data: filters

Allow users to specify which data fields they need.

```
GET /repos/:owner/:repo/comments
[
  {
    "html_url": "https://github.com/octocat/Hello-World/commit/6...",
    "url": "https://api.github.com/repos/octocat/Hello-World/comments/1",
    "id": 1,
    "node_id": "MDEzOkNvbW1pdENvbW1lbnQx",
    "body": "Great stuff",
    "path": "file1.txt",
    "position": 4,
    "line": 14,
    "commit_id": "6dcb09b5b57875f
    "user": {
    },
    "created_at": "2011-04-14T16:
    "updated_at": "2011-04-14T16:00:49Z"
  }
]
```


```
{
  organization(login: :owner) {
    name
    url
    repository(name: :repo) {
      name
      pullRequests(last: 10, states: [OPEN]) {
        edges {
          node {
            title
            comments(last: 10) {
              edges {
                node {
                  author {
                    name
                  }
                }...
```

Limit data: partial blobs

Binary Large OBjects can be large - let users get them in chunks.

Stateful: create sessions for user to stream data

Stateless: enable partial transfers so user can request chunks

Example: use HTTP HEAD and GET Accept-Range headers

Limit load: error codes

Good response codes let users know when to retry automatically (with backoff!) and when they need to adjust their request.

Examples:

Server errors:

Throttled

5xx HTTP codes:

500 Internal Service Error

Client errors:

Bad Precondition

4xx HTTP codes

401 Unauthorized

Limit load: bulk operations

Allow users to do a lot of work in a single request.

Examples:

- Get 20 specific users by id
- Update 20 specific users

Limit load: prioritize

Know the different use cases that users have - some requests are latency and failure sensitive and others are not.

Examples:

- Run a separate batch mode endpoint to isolate background job traffic from sensitive user traffic.
- Determine how “important” traffic is to give it a specific priority

Limiting load: quota

Introduce quotas to limit the amount of data or requests that users can have by default. Have a way to enable quota increases for users who need it (and test your system with the new limits!)

Examples:

- 10,000 requests per day per user
- 100 projects per account
- 100 objects per project

Limit load: rate limit

Limit the rate at which users can send requests. Introduce load shedding so that your system can reject requests when it’s overloaded.

Example:

Users can do a max of 10 qps

System returns a ‘throttled’ response

Limit load: caching

Make it easy for users to cache some data.

Example:

> HTTP headers for cache TTL and “ETag” to determine if the data has changed

Limit load: event-driven APIs

Make your API event-driven. Your API notifies users when relevant events happen rather than making them poll to check whether events have happened.

Trade-off: Instead of clients polling you frequently to see whether data has changed, your API needs to do that work internally.

Example:

> RESTHooks. Client subscribes by invoking your API with a callback URL, your system invokes that callback URL when a change happens.

Change management: lightweight SDKs

Some developers make an SDK to make it easier for users to access the API.

Tradeoff: Changing code is harder - it’s easier to fix bugs and release your API code than to get users to use a new SDK version.

Also, follow all your best practices (such as backoffs when retrying failures) in your SDK!

END