Social media scheduling API cheat sheet

Native scheduling, what to prepare and how long connections last, per network · checked against official docs, November 2026 · sources in social-media-scheduling-rules.json

1. Can the network schedule the post for you?

Network	Native	How / what your scheduler does	Prepare ahead
Facebook Page	Yes	`published=false` + `scheduled_publish_time`; 10 min to 30 days (guide; the feed reference says 75)	Further ahead: keep it in your queue, hand over inside the window
YouTube	Yes	`privacyStatus=private` + `publishAt` (ISO 8601)	Upload the video when scheduling; 100 uploads a day by default
Instagram	No	Container, then `media_publish` at publish time	Create the container minutes before: it expires after 24 h
Threads	No	Container, wait ~30 s, then `threads_publish`	Same 24 h container expiry
LinkedIn	No	`POST /rest/posts`; only `PUBLISHED` on creation	Upload media a few minutes before
TikTok	No	`creator_info`, then Direct Post init	User picks privacy level and consents when scheduling
X	No	`POST /2/tweets` at publish time	Upload media shortly before
Bluesky	No	`createRecord`; `createdAt` = current time	Upload blobs shortly before

2. Will the connection still work at publish time?

Network	Access token	Renewal
Facebook Page	Page token from a long-lived user token: no expiry date	Can still be invalidated: check it before publishing
Instagram	Long-lived user token: about 60 days	Reconnect or refresh before it ends
Threads	Long-lived token: 60 days	Refresh when at least 24 h old; dead if not refreshed in 60 days
LinkedIn	60 days	Refresh tokens 365 days (approved partners), not extended
TikTok	2 hours	Refresh token 180 days; a new one on every refresh
X	2 hours	Refresh token only with the `offline.access` scope
YouTube	Short-lived	Refresh token: 7 days if the Google app is in Testing; dies after 6 months unused

3. Six rules for your own scheduler

1. **Store the wall time and the IANA time zone** (`2027-03-15T09:00` + `America/New_York`), not today's offset. Convert to UTC at run time.
2. **Reject local times that do not exist** (clocks jump forward) and **pick one rule for times that happen twice**.
3. **Claim each job atomically** with a lease, so two workers never publish the same post.
4. **Keep an idempotency key per post and network**; after a timeout, look for the post on the network before retrying.
5. **Check content and connection at scheduling time and again before publishing**; warn the user days ahead.
6. **Store one result per network**, and retry only the networks that failed.

Free to use: data CC BY 4.0, code MIT · apiwings.com/blog/social-media-scheduling-api