ChatGPT

Production-Grade Guide to Supercharging Claudbot for Automated Crypto Trading on Coinbase in New York

Executive summary

A “supercharged” Claudbot is not one with more indicators—it’s one with **institution-grade plumbing**: reliable Coinbase connectivity, deterministic order execution, hard risk guardrails, end-to-end observability, and incident-ready operations. Coinbase’s **Advanced Trade API** is designed for programmatic trading via **REST + WebSocket**, with an official Python SDK that handles authentication and provides both REST and WebSocket clients. [1]

For **New York**, the engineering implications are mostly **constraint management** rather than “special endpoints”: plan for **product/network limitations** (e.g., Coinbase explicitly notes that multi-network support for certain assets is not available in NY), build your bot so it can gracefully refuse unsupported actions, and keep your architecture compliant-by-default. [2]

The largest performance and safety gains typically come from: - **Idempotent order placement** using `client_order_id` + local de-dupe locks (stops “phantom trades” and duplicated retries). [3]
- **A proper execution state machine** driven by **User Order Data WebSocket** updates, not “guessing” based on REST responses. [4]
- **Hard risk caps** enforced *before* sending orders (max position, max daily loss, circuit breakers, event lockouts), not advisory warnings after-the-fact.
- Production ops: **metrics + dashboards + alerting**, plus hardened key management and a tested “kill switch.” [5]

Assumptions and scope boundaries

Assumptions (explicit because Claudbot is unspecified): - **Language**: Python is assumed because Coinbase’s official Advanced Trade SDK in Python is the most direct route and provides both REST and WebSocket clients. If Claudbot is not Python-based, treat the code snippets as reference logic and port accordingly. [6]
- **Venue**: Coinbase spot trading via **Advanced Trade API** (REST + WebSocket endpoints). [7]
- **Infrastructure**: Linux VPS (or cloud VM), Docker for services, systemd for process supervision, Postgres as the system of record, Redis for short-lived locks/queues. [8]
- **NY constraints**: You should assume additional product/network constraints can apply. Coinbase’s help center explicitly states that “Support for multiple networks isn’t available in New York” for assets-on-multiple-networks behavior. [9]
- **Goal**: engineering excellence (robustness, correctness, risk control), not “alpha creation.”

Context implication: your earlier alerts (sell spam/phantom trades, risk stacking) are classic symptoms of missing **idempotency, state reconciliation, and hard pre-trade risk checks**—not primarily "bad data." [10]

Claudbot architecture inventory and reference design

Architecture inventory checklist

Because Claudbot is unspecified, inventory *what exists* before adding features. A production-grade bot typically has these components (verify each is present, correct, and testable):

- **Configuration & secrets**: environment-driven config, per-environment profiles (dev/paper/live), and no secrets in git. Coinbase explicitly recommends not embedding keys in code and keeping key files out of the source tree. [11]
- **Market data layer**: WebSocket ingestion + REST backfill, dedupe by sequence/time, and a stable internal "bar builder" (candles) plus optional order-book builder. Advanced Trade WebSocket provides channels like `candles`, `level2`, `ticker`, and `market_trades`. [12]
- **Signal/strategy layer**: deterministic, versioned strategies; no side effects (no direct order placement inside indicator code).
- **Risk engine (hard gate)**: centralized risk policy that can veto orders.
- **Execution engine**: order router, retries/backoff, slippage controls, idempotency, state machine. Advanced Trade orders are created via `/api/v3/brokerage/orders` with `client_order_id` and an `order_configuration` object describing the order type. [13]
- **Reconciliation loop**: periodic REST reconciliation (accounts/positions/orders) + event-driven updates from User Order Data WebSocket. [14]
- **Persistence**: immutable event log + normalized tables (orders, fills, positions, balances, risk snapshots). Prefer Postgres for durability (WAL is the durability mechanism). [15]
- **Observability**: structured logs, Prometheus metrics endpoint, Grafana dashboards. Grafana provisioning supports config-as-code. [16]
- **Ops controls**: admin commands (/pause, /resume), kill switch, incident runbooks. Coinbase security best practices recommend monitoring usage, logging/auditing usage, and having an incident response plan. [17]

Reference architecture

```
flowchart LR
  subgraph Data[Market Data Plane]
    WSMD[WS Market Data\n(ticker/level2/candles)] --> Dedupe[Dedupe & Sequence
Check]
    RESTMD[REST Backfill\n(candles/book/ticker)] --> Dedupe
    Dedupe --> StoreMD[(Time-series / Cache)]
  end

  subgraph Core[Trading Core]
    StoreMD --> Strategy[Strategy Engine\n(signals)]
    Strategy --> Risk[Risk Engine\n(hard veto + sizing)]
```

```
    Risk --> Exec[Execution Engine\n(idempotent + retries)]
    Exec --> Coinbase[Coinbase Advanced Trade\nREST Orders]
    WSUO[WS User Order Data\n(fills/status)] --> Reconcile[State Reconciler]
    Coinbase --> Reconcile
    Reconcile --> Journal[(Postgres\norders/fills/positions)]
  end

  subgraph Ops[Ops Plane]
    Journal --> Metrics[Metrics Exporter\nPrometheus]
    Metrics --> Dash[Dashboards\nGrafana]
    Journal --> Alerts[Alerts\n(Telegram/etc)]
    Logs[Structured Logs] --> Alerts
  end
```

This design deliberately separates the “what” (strategy) from the “may we” (risk) and the “how” (execution). That separation is the difference between a bot that *trades* and a bot that *survives*.

Coinbase Advanced Trade in New York

Authentication and key lifecycle

Coinbase Advanced Trade uses **CDP Secret API Keys** with JWT-based authentication for server-side apps and automated trading systems. [18]

Key algorithm nuance: - Coinbase recommends **Ed25519** generally, but notes **ECDSA is required** for the Coinbase App SDK / Advanced Trade SDK compatibility. [19]
- The Coinbase App API key authentication guide explicitly instructs selecting **ECDSA** and (optionally) setting an IP allowlist, portfolio restrictions, and permission restrictions. [20]

Security requirements in practice: - Never embed keys in code; store them outside the repo (env vars or a secrets manager). [11]
- Rotate keys regularly; Coinbase’s guidance for API key rotation recommends periodic rotation (they suggest 90–180 days). [21]

REST endpoints you’ll actually use

Advanced Trade API base URL: - `https://api.coinbase.com/api/v3/brokerage/{resource}` [22]

Coinbase lists required API-key permissions per endpoint (examples): - Accounts: `GET /accounts` requires `view` - Orders: `POST /orders` requires `trade` - Market data: `GET /products/{product_id}/candles` requires `view` [22]

Order creation: - `POST https://api.coinbase.com/api/v3/brokerage/orders` - Requires `client_order_id`, `product_id`, `side`, and an `order_configuration` describing the order type. [13]

From Coinbase's Create Order reference, `order_configuration` supports multiple modes such as market (IOC/FOK), limit (GTC/GTD/FOK), and time-weighted orders (TWAP). [23]

Execution guardrails: - The official Python SDK explicitly warns that auto-generating a `client_order_id` by passing an empty string removes the safeguard against accidentally placing duplicate orders. [24]

WebSocket endpoints, channels, and reliability rules

Advanced Trade WebSocket provides two production endpoints: - Market data: `wss://advanced-trade-ws.coinbase.com` - User order data: `wss://advanced-trade-ws-user.coinbase.com` [25]

Coinbase recommends using the market data endpoint as failover if user-order-data is your primary connection. [26]

Channels (Market Data feed) include: - `heartbeats`, `candles`, `ticker`, `ticker_batch`, `level2`, `market_trades`, and `status` [27]
Operational detail that matters: many channels can close within **60–90 seconds** without updates; Coinbase recommends subscribing to `heartbeats` to keep subscriptions open. [27]

Authentication nuance: - Coinbase's WebSocket authentication guide notes WebSocket JWTs are not built with a request method/path like REST JWTs. [28]
- Coinbase also states "For the most reliable connection, authenticate with a CDP API key when subscribing to any channel." [27]

Rate limits and CDP subscription tiers

Advanced Trade WebSocket rate limits (documented): - Connections: 8 per second per IP - Unauthenticated messages: 8 per second per IP [29]

Coinbase Exchange WebSocket subscription tiers (market data) are explicitly published (this is separate from Advanced Trade, but relevant if you consume Exchange market data): - Default: 10 subscriptions (free) - Paid tiers raise limits to 15 / 25 / 50 / 100 / 250 / 500 subscriptions with monthly costs shown in Coinbase's help center, and are managed via a CDP "Exchange data subscription" UI. [30]

CDP product pricing is described as "pay only for what you use" on Coinbase's CDP pricing page; treat it as the macro billing model, while Exchange market-data subscription tiers are a concrete example of a per-capability add-on. [31]

New York constraints worth engineering for

New York's crypto environment is structurally stricter (BitLicense/trust charter regime), which is why the list of fully-operational venues is smaller. [32]

A concrete NY constraint from Coinbase: - Coinbase says multi-network support for certain assets "isn't available in New York." Practically, this means your bot should not assume you can select cheaper/faster networks for transfers; design transfer logic as "single-network unless proven otherwise." [9]

Market data and execution engineering

Connectors: CCXT vs native Coinbase SDK

Even if you're Coinbase-only today, connector choice affects debuggability and time-to-production.

Connector comparison

Option	What it is	Strengths	Risks / gotchas	Best fit
Native official SDK (**coinbase-advanced-py** [33])	Coinbase-maintained Python SDK with REST + WS clients	Easiest path; built-in JWT helper; supports WS + auto reconnect; warns on idempotency pitfalls	Coinbase-only; SDK changes track Coinbase updates	Coinbase-only bots that want correctness fast [34]
CCXT [35]	Multi-exchange unified trading/ market-data interface	Swap venues later; consistent abstractions across exchanges	Abstraction leaks; Coinbase-specific features may be harder; you still must implement idempotency & risk	Multi-venue roadmap or benchmark harness [36]
Native HTTP (your own client)	Your own REST/WS integration + JWT generation	Maximum control; minimal dependencies	Highest engineering cost; easiest to get auth/rate-limit wrong	Only when SDK/ CCXT can't meet requirements [37]

A pragmatic production pattern is: **start with the official SDK**, put your own stable internal interface in front of it ("BrokerAdapter"), and only introduce CCXT if/when you truly need multi-venue portability. [38]

Market-data design: candles, order book, snapshots, and on-chain feeds

Advanced Trade gives you enough to build a robust internal market-data service: - WebSocket `candles` for real-time OHLC updates - WebSocket `level2` for an order-book snapshot + incremental updates - WebSocket `market_trades` / `ticker` / `ticker_batch` for trade/price updates [27]

Design principle: treat *WebSocket as primary* and *REST as reconciliation/backfill*. Coinbase itself emphasizes you are responsible for reading the message stream and using relevant messages (e.g., order books/ trades). [39]

Feed options comparison

Feed type	Coinbase source	Use	Typical failure mode	Mitigation
Candles	Advanced Trade WS `candles`	Strategy signals, bar-close logic	dropped WS connection / silent stalls	heartbeat subscription + reconnect; REST backfill candles [40]
Order book	Advanced Trade WS `level2`	slippage estimates, spread filters, microstructure	desync after missed messages	periodic snapshot rebuild; strict sequence/time checks [12]
User fill/order events	Advanced Trade WS User Order Data	execution state machine	missed fill events = wrong positions	reconcile with REST `List Orders` / `List Fills` [14]
Exchange market data (optional)	Exchange WebSocket feed	alternate market data plane	subscription caps / tiering	buy subscriptions if needed; batch subscriptions per message [30]
On-chain analytics (optional)	CDP SQL API / Onchain Data	regime filters, flow analysis	expensive/slow queries	cache + freshness controls; don't gate orders on it [41]
External on-chain providers (optional)	**Dune** [42], **Glassnode** [43], **DefiLlama** [44]	risk overlays, macro filters	quotas/rate limits	async fetch, local cache, degrade gracefully [45]

Execution engine: order types, idempotency, retries, slippage, state machine

Order types and time-in-force

Coinbase's Advanced Trade order management guide defines fulfillment policies like: - GTC (Good Till Canceled) - GTD (Good Till Date) - IOC (Immediate Or Cancel) [46]

The Create Order API shows multiple configuration structs for market and limit variants (e.g., market IOC/FOK; limit GTC/GTD/FOK; TWAP). [23]

Idempotency and duplicate prevention

Your previous "phantom trades" warning is what happens when retries place new orders unintentionally. Fix it systematically:

- Use a **deterministic** `client_order_id` per "intent" (e.g., `{strategy}-{symbol}-{timestamp}-{signal_hash}`).
- Persist it before sending the order.

- Treat any retry as a replay of the same intent.

Coinbase's API reference explains the concept of idempotency: repeated requests should produce the same result, especially for state-changing operations. [47]
Coinbase's Create Order endpoint requires `client_order_id`, which gives you the handle to implement this policy. [23]
Coinbase's official Python SDK explicitly warns that auto-generating IDs removes this duplicate-order safeguard. [24]

Retry and backoff

The official SDK's WebSocket client includes automatic reconnection using exponential backoff with a max number of retries (documented in the SDK README). [24]
Coinbase documentation also recommends exponential backoff for rate-limit handling in general. [48]

For REST, treat retries as: - Safe for **GET** (idempotent read) - **Dangerous for POST /orders** unless you enforce idempotency keys (`client_order_id`) and local locking. [49]

Execution state machine (non-optional)

Use WebSocket User Order Data as your "truth stream" for: - accepted/rejected - open/filled/partially filled/canceled - fills (execution price/size) [25]

```
stateDiagram-v2
  [*] --> IntentCreated
  IntentCreated --> RiskApproved : risk_ok
  IntentCreated --> Blocked : risk_veto

  RiskApproved --> SentToExchange : post_order
  SentToExchange --> Accepted : ws_order_ack
  SentToExchange --> Rejected : ws_reject

  Accepted --> PartiallyFilled : ws_fill_partial
  Accepted --> Filled : ws_fill_full
  Accepted --> Canceled : cancel_ack

  PartiallyFilled --> Filled : ws_fill_full
  PartiallyFilled --> Canceled : cancel_ack

  Rejected --> [*]
  Blocked --> [*]
  Filled --> PositionUpdated
  Canceled --> PositionUpdated
  PositionUpdated --> [*]
```

The key operational rule: **positions are derived from fills**, not from your strategy's "expected" outcome.

Risk management module: hard caps, circuit breakers, lockouts

A production-grade risk module is a centralized gate in front of execution. Use it to stop exactly the failure modes your alerts indicate (over-stacking, overbought chasing, spammy entries).

Risk policy template (YAML example)

```yaml
mode: live  # dev | paper | live
portfolio:
  max_gross_notional_usd: 5000
  max_net_notional_usd: 2500
  max_daily_loss_usd: 150
  max_drawdown_usd: 300
  max_orders_per_minute: 12

per_symbol:
  BTC-USD:
    max_position_notional_usd: 1000
    max_position_units: null
    max_leverage: 1
    cooldown_seconds: 120
    max_slippage_bps: 15
  ETH-USD:
    max_position_notional_usd: 800
    cooldown_seconds: 120
    max_slippage_bps: 20

circuit_breakers:
  consecutive_rejects: 5
  consecutive_losses: 4
  websocket_disconnect_seconds: 15
  api_429_per_minute: 20
  spread_widening_bps: 50

event_lockouts:
  enabled: true
  windows:
    - name: "major_macro_release"
      start_utc: "13:20"
      end_utc: "14:10"
      days: ["mon","tue","wed","thu","fri"]

allowlist:
  products: ["BTC-USD","ETH-USD"]
  order_types: ["market_ioc","limit_gtc"]
```

Implementation notes: - Enforce *both* portfolio-wide and per-symbol caps. - Add a per-symbol **cooldown** to prevent "sell spam." - Implement a global kill switch that flips bot mode to `paused` if circuit breakers trip.

Validation: backtesting, paper trading, and testing strategy

Backtesting frameworks comparison

Framework	Core strength	What it's good at	What it's bad at	Best use
vectorbt [50]	Very fast vectorized research; operates on pandas/NumPy; accelerated by Numba	Large parameter sweeps, regime research	Harder to simulate exchange microstructure precisely	Fast iteration and idea screening [51]
Backtrader [52]	Event-driven backtesting + live trading hooks	More realistic order/ commission modeling	More code; slower for large sweeps	Execution realism and strategy prototyping [53]
Backtesting.py [54]	Simple backtesting API	Quick toy models	Limited realism at scale	Teaching, quick sanity checks [55]

Data hygiene rules that prevent self-deception

Treat these as non-negotiable engineering requirements: - **No look-ahead**: indicators must only use information available at the decision timestamp. - **Bar alignment**: if you trade on 1-minute closes, you must decide after the close, not during. - **Fees and slippage**: include them; Advanced Trade has volume-based fees (and your fill price deviates from signals under volatility). [56]

Paper trading and sandbox strategy

Coinbase Advanced Trade "sandbox" is **static** and primarily useful for validating request/response shapes, not true market fills: - Sandbox endpoint: `https://api-sandbox.coinbase.com/api/v3/brokerage/{resource}` - Responses are mocked, static, and pre-defined; some endpoints vary with an `X-Sandbox:` header. [57]

Practical approach: - Use **Advanced Trade sandbox** to validate: auth wiring, serialization, schema parsing, idempotency handling code paths. - Use **paper/live shadow mode** in production code: read real market data, generate signals, run risk checks, but send orders to a "dry-run broker" that records what would have happened. - Graduate to tiny-size live trading only after reconciliation and kill-switches are proven.

Operations: observability, security, infrastructure, incident playbooks, and roadmap

Observability and alerting

Minimum viable production observability: - **Structured logs** (JSON) with correlation IDs per order intent. - **Metrics endpoint** scraped by Prometheus; Prometheus client libraries are the standard way to instrument services, exposing metrics over HTTP. [58]
- **Dashboards** in Grafana; provisioning supports configuration-as-code (ideal for GitOps). [59]
- **Error tracking** with **Sentry** [60] (exceptions, traces, alerting). [61]

Example Prometheus metrics you should track: - orders_submitted_total{product,side,type} - orders_rejected_total{reason} - fills_total{product} - slippage_bps_histogram{product} - ws_disconnects_total{endpoint} - risk_veto_total{rule} - api_429_total{endpoint}

Security and infrastructure baseline

Key management: - Follow Coinbase guidance: never embed keys in code; keep key files outside the repo; prefer env vars or a secrets manager; log and audit usage; have an incident response plan. [62]
- Use IP allowlisting when creating keys (Coinbase calls it recommended). [20]
- Rotate keys periodically per Coinbase guidance. [21]

Persistence choices: - Postgres as the system of record: WAL ensures data integrity by logging changes before data files are written, enabling crash recovery. [15]
- Redis for ephemeral locks/queues: Redis supports persistence via snapshots (RDB) and append-only files (AOF), configurable independently or together. [63]

Deployment: - Docker Compose to run bot + db + redis + monitoring in one reproducible stack (services/networks/volumes are first-class). [64]
- systemd for self-healing processes (restart policies) and controlled rollouts. [65]

Step-by-step migration checklist to a production-grade Coinbase integration

This checklist assumes you're migrating from "public Coinbase data" or legacy integrations into Advanced Trade SDK + WS user stream.

Step one: inventory current Claudbot 1. Identify your current data source(s): REST candles? WebSocket ticker? (Document symbol, interval, latency assumptions.) 2. Identify how orders are placed today: REST endpoint, library, or manual UI? 3. Confirm whether you have idempotency now (most bots don't).

Step two: create and secure CDP API keys 1. Create a **Secret API Key** in CDP. 2. Apply IP allowlist (if possible), portfolio restriction, and minimum permissions (`view` + `trade` only). [66]
3. Store secrets outside code; Coinbase explicitly warns against storing secrets in code. [67]

Step three: implement Advanced Trade REST via official SDK - Use `RESTClient` (official) and enable `rate_limit_headers=True` to surface rate-limit data in responses (per SDK docs). [68]

Step four: implement User Order Data WebSocket - Use the SDK's WebSocket user client (documented) so your bot's state machine is driven by exchange events. [69]

Step five: enforce idempotency and risk gates - Require `client_order_id` for every order; never auto-generate in production. - Put all risk checks before execution.

Step six: persistence and reconciliation - Persist every intent, every order ack, and every fill to Postgres. - Reconcile periodically with REST List Orders/List Fills endpoints. [70]

Sample Python snippets using the official SDK

Install and set environment variables (example from SDK README). [24]

```
pip3 install coinbase-advanced-py
export COINBASE_API_KEY="organizations/{org_id}/apiKeys/{key_id}"
export COINBASE_API_SECRET="-----BEGIN EC PRIVATE KEY-----
\nYOUR_PRIVATE_KEY\n-----END EC PRIVATE KEY-----\n"
```

Minimal REST client and account check (SDK supports `RESTClient`). [34]

```
from coinbase.rest import RESTClient
from json import dumps

client = RESTClient(rate_limit_headers=True)  # uses env vars; adds rate limit
headers to response objects

accounts = client.get_accounts()
print(dumps(accounts.to_dict(), indent=2))
```

Create a market buy with a deliberate `client_order_id` (SDK exposes `market_order_buy`). [24]

```
from coinbase.rest import RESTClient
from json import dumps
import uuid

client = RESTClient()

client_order_id = f"claudbot-{uuid.uuid4()}"  # better: deterministic per intent
order = client.market_order_buy(
    client_order_id=client_order_id,
    product_id="BTC-USD",
    quote_size="25"  # $25 notional
```

```
)
print(dumps(order.to_dict(), indent=2))
```

Subscribe to WebSocket market data with heartbeats (keep channels open). [40]

```
import time
from coinbase.websocket import WSClient

def on_message(msg):
    print(msg)

ws = WSClient(on_message=on_message)
ws.open()
ws.subscribe(product_ids=["BTC-USD"], channels=["ticker", "heartbeats"])
time.sleep(30)
ws.close()
```

Listen for authenticated user order updates via `WSUserClient` (SDK documents this). [69]

```
from coinbase.websocket import WSUserClient

def on_message(msg):
    print(msg)

ws_user = WSUserClient(on_message=on_message)  # uses env vars for keys
ws_user.open()
ws_user.subscribe(product_ids=["BTC-USD"], channels=["user", "heartbeats"])
ws_user.run_forever_with_exception_check()
```

Incident playbooks

Playbook: duplicate orders / phantom trades - Symptoms: multiple orders for same signal; “sell spam.” - Immediate response: pause trading; cancel open orders; set bot to “risk-only mode.” - Root cause checklist: - missing `client_order_id` or auto-generated per retry [71]
- retries on POST without dedupe - WebSocket disconnect causing “unknown state” and re-send - Fix: deterministic intent IDs + persistent intent log + lock per (strategy, product). [49]

Playbook: WebSocket silent stall - Symptoms: no ticks; stale candles; strategy still firing. - Immediate response: trigger circuit breaker; pause new entries. - Fix: - subscribe to heartbeats; Coinbase notes channels may close without updates. [27]
- use SDK reconnection or implement reconnect loops. [24]
- reconcile via REST backfill.

Playbook: rate-limit storm (429s) - Symptoms: REST failures, delayed reconciliations. - Response: exponential backoff; reduce polling; rely more on WS. - Note: Coinbase documents 429 on rate limiting and recommends backoff patterns. [72]

Prioritized implementation roadmap with milestones

A realistic “production-grade” rollout is staged. The timelines below assume a solo developer or small team; compress if you have dedicated engineering time.

```
gantt
  title Claudbot production roadmap
  dateFormat  YYYY-MM-DD
  axisFormat  %b %d

  section Foundations
  Repo structure + config profiles        :a1, 2026-02-24, 5d
  Secrets + key rotation procedure        :a2, 2026-02-24, 5d

  section Coinbase integration
  Advanced Trade REST via official SDK    :b1, 2026-03-03, 4d
  WS market data + heartbeats             :b2, 2026-03-03, 4d
  WS user order data + reconciliation     :b3, 2026-03-07, 5d

  section Execution and risk
  Idempotency + locks + state machine     :c1, 2026-03-12, 7d
  Risk engine + circuit breakers          :c2, 2026-03-12, 7d

  section Validation
  Backtest harness + data hygiene         :d1, 2026-03-19, 7d
  Paper/shadow mode                       :d2, 2026-03-19, 7d

  section Operations
  Metrics + dashboards + alerting         :e1, 2026-03-26, 6d
  Incident runbooks + game days           :e2, 2026-03-26, 6d

  section Launch
  Tiny-size live + graduated caps         :f1, 2026-04-02, 10d
```

Milestones (what “done” means): - **Foundation complete**: no secrets in repo; environment-based config; documented key rotation. [73]
- **Connectivity complete**: REST + both WS endpoints working; heartbeats subscribed; reconnect tested. [4]
- **Execution complete**: idempotent order placement; duplicates impossible by design. [74]
- **Risk complete**: hard caps enforced pre-trade; circuit breaker stops trading automatically.
- **Ops complete**: dashboards reflect trading health; alerts fire on critical faults. [16]

Final go-live checklist

Connectivity and correctness: - REST can list accounts and place/cancel orders with correct permissions. [22]
- WebSocket market data + heartbeats stable; reconnect tested. [40]
- WebSocket user order data drives order lifecycle; REST reconciliation agrees with WS. [14]

Execution safety: - Every order uses deterministic `client_order_id` (no empty string in prod). [75]
- Retry logic cannot create duplicate orders (locks + intent log). [47]

Risk: - Max position size, max daily loss, max order rate enforced locally. - Kill switch tested (manual and automatic triggers).

Ops: - Metrics endpoint scraped; dashboards deployed via provisioning. [16]
- Alerts configured for: WS disconnect, 429 spikes, consecutive rejects, drawdown breach. [17]

NY constraints: - Transfer/network logic does not assume multi-network availability in New York. [9]

Reference URLs

(Provided as plain URLs inside a code block per your “add links” request.)

```
Coinbase Advanced Trade overview:
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/overview

Advanced Trade REST endpoints (base URL + permissions):
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/rest-api

Create Order endpoint reference:
https://docs.cdp.coinbase.com/api-reference/advanced-trade-api/rest-api/orders/
create-order

Advanced Trade WebSocket overview (endpoints):
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/
websocket-overview

Advanced Trade WebSocket channels:
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/
websocket-channels

Advanced Trade WebSocket rate limits:
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/
websocket-rate-limits

Coinbase App API key authentication (ECDSA requirement):
https://docs.cdp.coinbase.com/coinbase-app/authentication-authorization/api-key-
authentication
```

```
Authentication overview (Ed25519 vs ECDSA compatibility matrix):
https://docs.cdp.coinbase.com/get-started/authentication/overview

Official coinbase-advanced-py SDK repo:
https://github.com/coinbase/coinbase-advanced-py

SDK docs:
https://coinbase.github.io/coinbase-advanced-py/

Advanced Trade API sandbox (static mocked responses):
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/sandbox

Exchange market data subscriptions + pricing tiers:
https://help.coinbase.com/en/exchange/managing-my-account/market-data-
connections

Exchange WebSocket rate limits:
https://docs.cdp.coinbase.com/exchange/websocket-feed/rate-limits

NY note on multi-network assets:
https://help.coinbase.com/en/coinbase/trading-and-funding/sending-or-receiving-
cryptocurrency/assets-on-multiple-networks

CCXT:
https://github.com/ccxt/ccxt

vectorbt:
https://vectorbt.dev/
https://github.com/polakowo/vectorbt

Backtrader:
https://www.backtrader.com/
https://github.com/mementum/backtrader

Backtesting.py:
https://kernc.github.io/backtesting.py/

TradingView webhooks:
https://www.tradingview.com/support/solutions/43000529348-how-to-configure-
webhook-alerts/

Prometheus:
https://prometheus.io/docs/instrumenting/clientlibs/
https://prometheus.github.io/client_python/instrumenting/

Grafana provisioning:
https://grafana.com/docs/grafana/latest/administration/provisioning/
```

```
Redis persistence:
https://redis.io/docs/latest/operate/oss_and_stack/management/persistence/

PostgreSQL WAL:
https://www.postgresql.org/docs/current/wal-intro.html

Sentry docs:
https://docs.sentry.io/

CDP SQL API:
https://docs.cdp.coinbase.com/data/sql-api/welcome

Dune API:
https://docs.dune.com/api-reference/overview/getting-started

Glassnode API credits/rate limits:
https://docs.glassnode.com/basic-api/api-credits

DefiLlama API docs:
https://api-docs.defillama.com/
```

---

1 7 33 Welcome to Advanced Trade API
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/overview?utm_source=chatgpt.com

2 9 Assets on multiple networks
https://help.coinbase.com/en/coinbase/trading-and-funding/sending-or-receiving-cryptocurrency/assets-on-multiple-networks?utm_source=chatgpt.com

3 13 23 54 74 Create Order - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/api-reference/advanced-trade-api/rest-api/orders/create-order

4 14 25 26 35 Advanced Trade WebSocket Overview
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/websocket-overview?utm_source=chatgpt.com

5 16 59 Provision Grafana | Grafana documentation
https://grafana.com/docs/grafana/latest/administration/provisioning/?utm_source=chatgpt.com

6 38 Python SDK Overview - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/sdk?utm_source=chatgpt.com

8 64 Docker Compose
https://docs.docker.com/compose/?utm_source=chatgpt.com

10 47 49 Idempotency - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/api-reference/v2/idempotency?utm_source=chatgpt.com

11 17 50 62 67 73 Best Practices: API Security
https://docs.cdp.coinbase.com/get-started/authentication/security-best-practices?utm_source=chatgpt.com

12 27 40 60 Advanced Trade WebSocket Channels
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/websocket-channels?utm_source=chatgpt.com

15 43 Documentation: 18: 28.3. Write-Ahead Logging (WAL)
https://www.postgresql.org/docs/current/wal-intro.html?utm_source=chatgpt.com

18 19 Overview - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/get-started/authentication/overview?utm_source=chatgpt.com

20 66 Coinbase App API Key Authentication
https://docs.cdp.coinbase.com/coinbase-app/authentication-authorization/api-key-authentication?utm_source=chatgpt.com

21 How to rotate your API key
https://help.coinbase.com/exchange/managing-my-account/how-to-rotate-your-api-key?utm_source=chatgpt.com

22 44 70 Advanced Trade API Endpoints - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/rest-api

24 34 42 52 68 69 71 75 raw.githubusercontent.com
https://raw.githubusercontent.com/coinbase/coinbase-advanced-py/master/README.md

28 Advanced Trade WebSocket Authentication
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/websocket-authentication?utm_source=chatgpt.com

29 39 Advanced Trade WebSocket Rate Limits
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/websocket/websocket-rate-limits?utm_source=chatgpt.com

30 Market Data Connections
https://help.coinbase.com/en/exchange/managing-my-account/market-data-connections?utm_source=chatgpt.com

31 CDP Pricing
https://www.coinbase.com/developer-platform/pricing?utm_source=chatgpt.com

32 Virtual Currency Business Licensing
https://www.dfs.ny.gov/virtual_currency_businesses?utm_source=chatgpt.com

36 ccxt/ccxt: A cryptocurrency trading API with more than 100 ...
https://github.com/ccxt/ccxt?utm_source=chatgpt.com

37 GitHub - coinbase/coinbase-advanced-py: The Advanced API Python SDK is a Python package that makes it easy to interact with the Coinbase Advanced API. The SDK handles authentication, HTTP connections, and provides helpful methods for interacting with the API.
https://docs.cdp.coinbase.com/sdks/advanced-trade/python

41 Welcome to SQL API - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/data/sql-api/welcome?utm_source=chatgpt.com

45 Authentication - Dune Docs
https://docs.dune.com/api-reference/overview/authentication?utm_source=chatgpt.com

46 Advanced API Order Management
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/guides/orders?utm_source=chatgpt.com

48 72 Rate Limits - Coinbase Developer Documentation
https://docs.cdp.coinbase.com/api-reference/v2/rate-limits?utm_source=chatgpt.com

51 VectorBT: Getting started
https://vectorbt.dev/?utm_source=chatgpt.com

53 mementum/backtrader: Python Backtesting library for ...
https://github.com/mementum/backtrader?utm_source=chatgpt.com

55 Backtesting.py - Backtest trading strategies in Python
https://kernc.github.io/backtesting.py/?utm_source=chatgpt.com

56 Advanced Trade API FAQ
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/faq?utm_source=chatgpt.com

57 Advanced Trade API Sandbox
https://docs.cdp.coinbase.com/coinbase-app/advanced-trade-apis/sandbox?utm_source=chatgpt.com

58 Client libraries
https://prometheus.io/docs/instrumenting/clientlibs/?utm_source=chatgpt.com

61 Sentry Docs | Application Performance Monitoring & Error ...
https://docs.sentry.io/?utm_source=chatgpt.com

63 Redis Persistence and Durability: RDB Snapshots & AOF
https://redis.io/tutorials/operate/redis-at-scale/persistence-and-durability/?utm_source=chatgpt.com

65 systemd.service(5) - Linux manual page
https://man7.org/linux/man-pages/man5/systemd.service.5.html?utm_source=chatgpt.com