



How to Trade with Magellan

A practical handbook for grid trading on SOL/USDC

Powered by **Jupiter**

magellan.money

A practical handbook for getting the most out of Magellan's grid trading strategy on SOL/USDC.

This guide assumes you understand Solana, token swaps, and basic DeFi concepts. It focuses on *how to think about the parameters* and *how to adapt to market conditions*, not on installation or configuration syntax (see README.md and CONFIGURATION.md for those).

Table of Contents

1. [How Magellan Makes Money](#)
2. [Understanding the Grid](#)
3. [The Cost of Trading](#)
4. [Tuning for Profit: The Core Parameters](#)
5. [Risk Parameters: Protecting Your Capital](#)
6. [Circuit Breaker: Stop-Loss Cascade Protection](#)
7. [Capital Sizing and Unit Allocation](#)
8. [Reading the Market: When to Use Which Settings](#)
9. [Worked Examples](#)
10. [The Paper-to-Live Pipeline](#)
11. [Common Mistakes](#)
12. [Advanced Tactics](#)
13. [Trading Tokenized Stocks](#)
14. [Quick Reference Cheat Sheet](#)

Appendices

- [Appendix A: Reading the SOL/USDC Operator Wallet panel](#)
 - [Appendix B: Withdrawing money to your beneficiary wallet](#)
 - [Appendix C: Selling a position by hand](#)
 - [Appendix D: Wind-Down Mode](#)
 - [Appendix E: Telegram Bot](#)
 - [Appendix F: How to Change Capital and Other Structural Parameters](#)
-

1. How Magellan Makes Money

Magellan is a **scalping grid trader**. It profits from the natural oscillation of SOL's price, the constant small ups and downs that happen even in a flat market.

The core cycle:

```
Price drops → Bot buys SOL cheap
Price recovers → Bot sells SOL at a small profit
Repeat
```

Each unit operates independently: buy low, sell high, recycle. How much a single trade earns depends on your `TAKE_PROFIT` setting: a setting just above the cost floor (1.7–1.9% in live mode, where the bot refuses 1.60% or less at the default `SLIPPAGE_BPS=50`) captures many small gains, while a higher setting (2.0–3%) earns more per trade but fires less often. Either way, across 10 units running 24/7, these gains compound.

A day in the life of Unit 3 (with \$50 per unit, `TAKE_PROFIT=2.0`, SOL at ~\$150):

```
09:14 UTC - SOL drops to $148.50 → Unit 3's buy target hit → buys 0.3367 SOL for $50
09:14 UTC - Unit 3 is now HOLDING, watching for $151.47 (+2.0% above $148.50)
11:42 UTC - SOL recovers to $151.47 → sell triggers → sells 0.3367 SOL for $51.00
11:42 UTC - Unit 3 earns $1.00 gross, ~$0.15 in fees → $0.85 net profit
11:42 UTC - Unit 3 returns to IDLE → immediately gets a new buy target below market
→ Ready to do it all over again
```

One unit, one cycle, \$0.85 profit. Multiply by 10 units, several cycles per day, 365 days a year.

Key insight: Magellan doesn't need SOL to go up. It needs SOL to *move*. A sideways market with frequent 2–4% oscillations is ideal. A straight line (up or down) is the enemy.

Why grid trading works on SOL/USDC

SOL/USDC is one of the most liquid pairs on Solana with tight spreads on Jupiter. SOL typically oscillates 1–5% intraday even on "quiet" days. That's exactly the range Magellan exploits:

- **High liquidity** → minimal slippage on Jupiter swaps
 - **Frequent oscillations** → more buy/sell cycles per day
 - **Deep liquidity pools** → reliable execution even for larger trades (Jupiter routes through Raydium, Orca, and other top Solana AMMs)
-

2. Understanding the Grid

The grid is the backbone of the strategy. Think of it as a ladder of buy orders placed below the current price.

How it works

With SOL at \$150 and these settings:

```
PRINCIPAL=500          # total USDC budget (USD, absolute amount)
SPLIT_NUMBER=10        # number of grid slots (integer, unit size = PRINCIPAL /
SPLIT_NUMBER)
PRINCIPAL_LEFTOVER=10  # reserve held back as buffer (% , percentage of principal)
GRID_SPREAD=0.2        # distance between each unit's buy level (% , below market
price)
TAKE_PROFIT=2.0        # gain above entry at which a unit sells (%)
```

The bot creates **9 active units** (10 total minus 1 buffer) with staggered buy targets:

Unit	Buy Target	Distance Below Market
1	\$149.70	-0.2%
2	\$149.40	-0.4%
3	\$149.10	-0.6%
4	\$148.80	-0.8%
5	\$148.50	-1.0%
6	\$148.20	-1.2%
7	\$147.90	-1.4%
8	\$147.60	-1.6%
9	\$147.30	-1.8%

When price drops to a unit's target, that unit buys. When price recovers by `TAKE_PROFIT%` above its entry, it sells.

The grid center

The grid center is the reference price from which all buy targets are calculated. It updates when:

- The bot starts fresh (uses current market price)
- A **grid reset** triggers (price drifted too far from center)

Units already holding SOL are never affected by grid resets. Only unfilled WAITING_TO_BUY orders get recalculated.

Example: grid center is \$150, `GRID_RESET_THRESHOLD=2` (2% drift tolerance). SOL rises to \$154 (+2.7% drift):

Before reset:	After reset:
Grid center: \$150	Grid center: \$154
Unit 1: WAITING at \$149.70 (stale) →	Unit 1: WAITING at \$153.69
Unit 2: WAITING at \$149.40 (stale) →	Unit 2: WAITING at \$153.38
Unit 3: HOLDING at \$149.10 entry →	Unit 3: HOLDING at \$149.10 (unchanged!)
Unit 4: WAITING at \$148.80 (stale) →	Unit 4: WAITING at \$153.08

Unit 3 was already holding SOL, so it keeps its position and its original take-profit target. Only the unfilled WAITING units get fresh targets near the new price.

Grid spread: wide vs. narrow

The deepest unit sits at `active_units × GRID_SPREAD` below the current price (since the first unit starts at `1 × GRID_SPREAD` below market):

GRID_SPREAD	Max Depth (9 units)	Needs STOP_LOSS of at least	Behavior
0.1%	0.9% below market	0.9%	Dense - many units trigger on small dips
0.3%	2.7% below market	2.7%	Balanced - needs a stop-loss like the Volatile preset's 3%
0.5%	4.5% below market	4.5%	Wide - catches deeper dips, fewer triggers
1.0%	9.0% below market	9.0%	Very wide - only triggers on significant drops

The bot refuses a grid deeper than `STOP_LOSS` (see `GRID_SPREAD` in Section 4), so at the default `STOP_LOSS=2` a 9-unit grid accepts a spread of at most 0.22%.

Narrow grids (0.1–0.2%) fire more often but risk having many units buy at nearly the same price. If the price keeps dropping, they all take losses together.

Wide grids (0.5–1.0%) catch different price levels, giving you better average entries during a dip. But in a tight oscillation range, only the top 1–2 units ever trigger, and on 9 active units they need a `STOP_LOSS` of 4.5–9%, wider than any preset in Section 8.

3. The Cost of Trading

Every trade has a cost. If your take-profit doesn't cover these costs, you lose money on winning trades.

Live mode costs

In live mode, round-trip costs come from:

Cost Component	Typical Range	Notes
DEX pool fees	0.05–0.15% per side	Charged by underlying AMMs (Raydium, Orca); Jupiter routes through them
Slippage	0.01–0.10% per side	Depends on trade size and pool depth
Solana tx fee	~0.000005 SOL	Negligible (~\$0.001)
Priority fees	0–0.0001 SOL	Only during congestion; set <code>PRIORITY_FEE=Y</code> and <code>PRIORITY_FEE_LAMPORTS</code> in config.env, or leave <code>PRIORITY_FEE=N</code> for Jupiter auto-estimation

Typical round-trip cost in live mode: 0.1–0.5%

That is what a round trip usually costs, not what the bot plans for. Its profit check assumes every fill hits your full `SLIPPAGE_BPS`, which at the default 50 bps puts the live round trip at **1.60%** (0.50% slippage plus 0.30% fees, on each leg), and it refuses a `TAKE_PROFIT` at or below that however cheaply your trades actually fill. This means, at `SLIPPAGE_BPS=50`:

- `TAKE_PROFIT=2.0` → accepted, net profit ~1.5–1.9% per trade at typical costs ✓
- `TAKE_PROFIT=1.8` → accepted, net profit ~1.3–1.7% per trade at typical costs ✓ (only 0.2% above the floor, so raising `SLIPPAGE_BPS` past 60 would get it refused)
- `TAKE_PROFIT=1.6` or lower → refused ✗

Paper mode

Paper mode runs the full strategy using real market prices but no real transactions. Every buy and sell is simulated. Your wallet is never touched. It is the right tool to validate a new config before risking real capital.

	Paper mode	Live mode
Trades	Simulated, no on-chain swap	Real Jupiter swaps on Solana
Wallet balance	Unchanged	Actual USDC/SOL moves
Fees and slippage	Estimated via config	Real DEX fees and real slippage
Risk	Zero	Real
Best for	Testing parameters, validating a new config	Actual trading

All grid parameters apply exactly as in live mode: `PRINCIPAL`, `SPLIT_NUMBER`, `TAKE_PROFIT`, `STOP_LOSS`, and so on. The two paper-specific parameters control simulated costs:

- `SLIPPAGE_BPS`: simulated slippage per trade, in basis points
- `PAPER_DEX_FEE`: simulated DEX fee per trade, as a decimal

```
# Per-leg cost (one swap):
slippage_pct    = SLIPPAGE_BPS / 2 / 10000    # half of slippage budget, converted from
bps to %
dex_fee_pct     = PAPER_DEX_FEE                # DEX fee as decimal (0.0025 = 0.25%)
cost_per_leg    = slippage_pct + dex_fee_pct

# Full cycle (buy + sell):
round_trip_cost = cost_per_leg * 2

# Example: SLIPPAGE_BPS=50, PAPER_DEX_FEE=0.0025
per leg        = 50/2/10000 + 0.0025 = 0.0025 + 0.0025 = 0.50%
round-trip     = 0.50% * 2 = 1.0%
```

Two practical presets:

Goal	SLIPPAGE_BPS	PAPER_DEX_FEE	Round-trip cost
Realistic test (matches live)	50	0.0025	~1.0%
Faster test (more trade activity)	10	0.0005	~0.2%

Use the realistic preset when validating a config you plan to deploy live. The deliberately pessimistic costs mean paper profits are harder to achieve, so you don't get false confidence. Use the faster preset when you want to observe bot behaviour across many trades quickly and don't need cost accuracy.

The profitability formula

```
Net profit per trade = TAKE_PROFIT% - round-trip cost%
```

Example with \$50 unit size in live mode:

- `TAKE_PROFIT=2.0%` , round-trip cost ~0.3%
- Net profit = $2.0\% - 0.3\% = 1.7\% \times \$50 = \text{\$0.85 per trade}$ ✓
- If you do 10 winning trades per day: **\\$8.50/day**

Same setup with `TAKE_PROFIT=0.3%` :

- Net profit = $0.3\% - 0.3\% = 0.0\% \times \$50 = \text{\$0.00 per trade}$ ✗
- You're working for free. Every trade is a coin flip after fees

Rule of thumb for live trading: clear the bot's floor with room to spare. The floor is its worst-case round-trip figure, **1.60%** at the default `SLIPPAGE_BPS=50` , and a `TAKE_PROFIT` at or below it is refused even if your real fills cost ~0.3%. `TAKE_PROFIT=2.0` clears it by 0.4% and is still more than 6× a typical ~0.3% round trip. This gives you a safety margin for slippage spikes.

Understanding round-trip cost

Round-trip cost is the total percentage you lose just by completing one full trade cycle: a buy followed by a sell. It is not a single fee, it is the sum of everything the market and the protocol take from you on both legs.

Why both sides? Because you pay costs when you buy SOL, and again when you sell it. Even if the price returns exactly to where you bought, you end up with less USDC than you started with.

Step	Amount
Start	\$100.00 USDC
Buy SOL (0.5% cost)	\$99.50 worth of SOL
Price returns to entry	SOL still worth \$99.50 in USDC terms
Sell SOL (0.5% cost)	\$99.00 USDC received
Round-trip loss	\$1.00 (1.0%)

At 1.0% round-trip cost, your `TAKE_PROFIT` needs to exceed 1.0% just to break even. That is why the rule of thumb above asks for room to spare: profit must meaningfully clear the cost floor, not just scrape past it.

Round-trip cost is your floor. Every parameter decision in Magellan starts here. Before tuning `TAKE_PROFIT`, `GRID_SPREAD`, or anything else, know what your round-trip cost actually is. Estimate it in paper mode first, then verify it against your first live trades.

The dashboard checks this for you

You do not have to hold the number in your head. When you change `TAKE_PROFIT`, `STOP_LOSS` or `SLIPPAGE_BPS` from the **Settings** panel, the dialog works out the cost for you. A `STOP_LOSS` below the cost of a single trade gets a warning banner on the confirmation screen. A `TAKE_PROFIT` at or below the round-trip cost is refused outright, with an error under the field, and Confirm does nothing until you change it:

Round-trip cost is 1.00% but TAKE_PROFIT is 0.9%. Every winning trade would close at a net loss. Raise TAKE_PROFIT above 1.00%, or lower SLIPPAGE_BPS.

Note that `SLIPPAGE_BPS` is in that list. Raising your slippage tolerance lifts the cost floor without you touching `TAKE_PROFIT` at all: going from 50 to 300 bps moves the live floor from 1.60% to 6.60%, which quietly turns a healthy 3% target into a losing one. The dialog catches that case too.

Read the figure as a ceiling, not a forecast. It assumes every single fill hits your maximum slippage, which on a liquid SOL/USDC pair almost never happens: `SLIPPAGE_BPS` is the worst price you are willing to accept, not the price you normally get. So a warning means "this leaves no margin for a bad fill", not "you are guaranteed to lose". A `TAKE_PROFIT` at or below this figure is refused, though, not just flagged: the dialog will not apply it, the dashboard server rejects it, and an SSH edit to `config.env` is refused at reload. While that value stays in the file every later reload is refused too, so nothing else you save in the meantime is applied, and a restart would start trading it. Raise `TAKE_PROFIT` or lower `SLIPPAGE_BPS` first.

4. Tuning for Profit: The Core Parameters

These five parameters determine how and when the bot trades.

TAKE_PROFIT: how much you make per trade

This is the percentage gain above entry price that triggers a sell.

```
TAKE_PROFIT=2.0    # Sell when +2.0% above buy price
```

Higher take-profit (2.0–3%):

- More profit per trade
- Fewer trades trigger (price must recover further)
- Better for volatile markets with large swings
- More resilient to transaction cost variance

Lower take-profit (1.7–1.9%):

- Less profit per trade
- More trades trigger (price doesn't need to recover as far)
- Better for choppy markets with tight oscillations
- Riskier, with thin margin for error on costs, and it has a hard floor: in live mode at the default

`SLIPPAGE_BPS=50` the bot refuses 1.60% or less

Example: SOL at \$150 with `TAKE_PROFIT=0.7`

- Unit buys at \$149.25
- Sell triggers at $\$149.25 \times 1.007 = \mathbf{\$150.29}$
- Price only needs to bounce \$1.04 for a win

Example: Same setup with `TAKE_PROFIT=0.3`

- Unit buys at \$149.25
- Sell triggers at $\$149.25 \times 1.003 = \mathbf{\$149.70}$
- Price only needs to bounce \$0.45, but costs eat more of the margin

STOP_LOSS: how much you're willing to lose per trade

This is the percentage drop below entry price that triggers a forced sell.

```
STOP_LOSS=2    # Cut losses at -2% below buy price
```

Stop-loss exists to **free up capital**. Without it, a unit that bought during a downturn sits idle holding a losing position forever, unable to participate in new trades.

Tight stop-loss (1–1.5%):

- Cuts losses quickly
- Frees units to re-enter at better prices
- But in a volatile market, normal oscillations can trigger stop-losses before the price recovers. This is called **getting stopped out**

Wide stop-loss (2–3%):

- Gives positions more room to recover
- Fewer false stop-outs from normal volatility
- But when a real downturn hits, you lose more per unit

Example, tight vs. wide stop-loss in the same scenario:

Unit buys at \$150. SOL drops to \$147 (-2%), then recovers to \$152 (+1.3%):

Setting	What happens	Result
STOP_LOSS=1.5	Stop-loss triggers at \$147.75 → forced sell → unit misses the recovery	-\$0.75 loss (1.5% of \$50 unit)
STOP_LOSS=2	Stop-loss triggers at \$147.00 → forced sell → unit misses the recovery	-\$1.00 loss (2% of \$50 unit)
STOP_LOSS=3	Price never hits \$145.50 → unit holds through the dip → sells at \$151.05 (+0.7%)	+\$0.35 profit

The wide stop-loss won here because the dip reversed. But if SOL had kept falling to \$140, the wide stop-loss would have lost \$1.50 (3%) instead of \$0.75 (1.5%).

The stop-loss / take-profit ratio matters, and costs make it worse. Trading costs (DEX fees + slippage) reduce your net profit on wins but *increase* your actual loss on stop-outs (you pay fees on the losing sell too). In live mode, with ~0.3% round-trip costs:

TAKE_PROFIT	STOP_LOSS	Net Win	Actual Loss	Wins to Recover	Min Win Rate
0.7%	2%	~0.4%	~2.3%	~6 wins	>85%
0.7%	3%	~0.4%	~3.3%	~8 wins	>89%
0.5%	2%	~0.2%	~2.3%	~12 wins	>92%
0.5%	1.5%	~0.2%	~1.8%	~9 wins	>90%

These numbers look demanding, but grid trading in oscillating markets naturally produces very high win rates (often 90–95%) because most buy-the-dip entries recover. The key is ensuring the rare stop-loss events don't wipe out accumulated gains.

The practical takeaway: wider TAKE_PROFIT (2.0%+ in live mode, clear of the bot's 1.60% floor at the default slippage) and moderate STOP_LOSS (2%) give you the best cushion. Avoid thin take-profits with wide stop-losses. The math is brutal.

GLOBAL_SL_COOLDOWN_MIN: pause ALL buys after any stop-loss

```
GLOBAL_SL_COOLDOWN_MIN=60    # Pause all buys for 60 minutes after any stop-loss
```

When any unit gets stopped out, it means the market has moved significantly against the strategy. Rather than letting other units continue buying into the decline, `GLOBAL_SL_COOLDOWN_MIN` pauses **all new buys across the entire bot** for the specified minutes. This is a bot-wide safety mechanism, not a per-unit cooldown.

If a second stop-loss fires during the pause, the timer resets from that point, extending the pause. Existing holdings are unaffected (they can still take profit or stop out). Only new buy orders are blocked.

Recommended values: `30 – 90` minutes for active trading. Use `0` only if you want maximum re-entry speed and accept the cascade risk.

GRID_SPREAD: how far apart your buy orders are

```
GRID_SPREAD=0.2    # 0.2% between each unit's buy target
```

This controls how deep your grid extends below the current price. The deepest unit sits at `GRID_SPREAD × active_units` below market (since the first unit starts at $1 \times \text{GRID_SPREAD}$).

With 9 active units and `GRID_SPREAD=0.2`, your deepest unit is 1.8% below the current price.

Key trade-off: wider spread = better diversified entries but fewer triggers.

The depth rule, which the bot enforces: the whole grid has to fit inside one unit's stop-loss, so `GRID_SPREAD * active_units` must be at or below `STOP_LOSS`. Put the other way, `GRID_SPREAD` can be at most `STOP_LOSS / active_units`. A deeper grid is refused by the Settings dialog and at reload, because the units that filled near the top would stop out before the deepest units ever filled, realizing losses in a decline instead of accumulating a position. `GRID_SPREAD=0` is exempt: it has no ladder at all.

Example: `STOP_LOSS=2` with 9 active units allows a spread of at most $2 / 9 \approx 0.22\%$, so use `GRID_SPREAD=0.2`. With `TAKE_PROFIT=2.0` and SOL at \$150:

- Unit 1 buys at \$149.70 (-0.2%), sell target: \$152.69
- Unit 2 buys at \$149.40 (-0.4%), sell target: \$152.39
- Unit 3 buys at \$149.10 (-0.6%), sell target: \$152.08
- The deepest unit (Unit 9) waits at \$147.30 (-1.8%), above the highest stop-loss (Unit 1's, at \$146.71), so a decline fills the whole ladder before any unit is stopped out
- SOL drops to \$148.90 (triggers Units 1-3), then recovers to \$152.69
- **All 3 units sell profitably** during the recovery, each at their own take-profit level

If `GRID_SPREAD` were too wide (e.g., 1%), it would fail twice. On 9 active units the ladder would be 9% deep, far past a 2% stop-loss, so the bot refuses it. And even with a stop-loss wide enough to allow it, Unit 2 would wait at \$147.00 and Unit 3 at \$145.50, so most units would never trigger in a small dip, leaving them idle.

GRID_RESET_THRESHOLD: when to rebuild the grid

```
GRID_RESET_THRESHOLD=2    # Reset grid if price drifts >2% from center
```

When the market moves significantly in one direction, your grid of unfilled buy orders becomes stale. They're all too far from the current price to ever trigger. The grid reset cancels them and rebuilds around the new price.

Low threshold (1–1.5%): resets frequently, keeps grid close to market. Risk: grid "chases" the price during trends, buying at each new level during a sustained drop.

High threshold (2–3%): only resets on significant moves. More stable, but if price slowly drifts away, the grid sits idle for longer.

Example, low vs. high threshold during a \$150 → \$155 trend:

Threshold	Resets at	What happens
1.5%	\$152.25 (+1.5%)	Grid rebuilds at \$152.25, then again at \$154.53. Constant chasing. If it's a downtrend disguised as noise, units keep buying higher and higher
3%	\$154.50 (+3%)	Grid stays anchored at \$150 until +3% drift. Fewer resets, but unfilled buy orders may sit stale for hours until the reset finally fires

In a genuine uptrend, low threshold = more opportunities. In a fakeout, low threshold = more exposure. There's no perfect answer, so match it to your risk tolerance.

Rule of thumb: set `GRID_RESET_THRESHOLD` to approximately your grid's max depth. If your deepest unit sits at 1.8% below market (9 units $\times$ 0.2% spread), a reset threshold of about 2% ensures the grid rebuilds before it becomes completely out of range. The bot's own requirement is looser but absolute: it refuses a `GRID_RESET_THRESHOLD` at or below `GRID_SPREAD`, because the grid would re-centre before price could reach even the first unit's target, and the bot would never buy.

5. Risk Parameters: Protecting Your Capital

MAX_DAILY_LOSS: the circuit breaker

```
MAX_DAILY_LOSS=5    # Halt if daily losses exceed 5% of PRINCIPAL
```

With `PRINCIPAL=500`, this means trading halts if you lose \$25 in a single day.

When triggered, the bot enters **sell-only mode**: it still executes stop-loss sells on holding positions (to limit further damage) but doesn't open any new buys. Resets at midnight UTC.

Level	MAX_DAILY_LOSS	With \$500 principal	With \$5,000 principal
Conservative	3%	Halts after -\$15	Halts after -\$150
Moderate	5%	Halts after -\$25	Halts after -\$250
Aggressive	10%	Halts after -\$50	Halts after -\$500

Example: `PRINCIPAL=500`, `SPLIT_NUMBER=10`, `STOP_LOSS=3`, `MAX_DAILY_LOSS=3`

Limit: 3% of \$500 = **\$15.00 max daily loss**. Each unit = \$50. A stop-loss hit costs $50 \times 3\% =$ **\$1.50 per unit**.

SOL flash-crashes -8% during the morning session:

Time	Event	Loss this stop	Cumulative loss
09:14	Unit 1 hits stop-loss at -3%	-\$1.50	-\$1.50
09:14	Unit 2 hits stop-loss at -3%	-\$1.50	-\$3.00
09:15	Unit 3 hits stop-loss at -3%	-\$1.50	-\$4.50
09:16	Grid resets at new lower price, 9 units redeploy		-\$4.50
09:22	Units 4–6 hit stop-loss (crash continues)	-\$4.50	-\$9.00
09:23	Units 7–9 hit stop-loss	-\$4.50	-\$13.50
09:24	Grid resets again, Unit 10 fills a new buy		-\$13.50
09:25	Unit 10 hits stop-loss, cumulative reaches -\$15.00	-\$1.50	-\$15.00
09:25	Circuit breaker fires. Bot enters sell-only mode.		

From this point the bot opens no new buys. Any remaining positions can still be sold via stop-loss, but no fresh capital is deployed. Trading resumes at **midnight UTC** when the daily counter resets.

Without `MAX_DAILY_LOSS` : the grid keeps resetting into the crash, buying at \$148, then \$142, then \$136... each reset deploying fresh capital into a continuing dump. Losses compound far beyond \$15.

PRINCIPAL_LEFTOVER: the safety buffer

```
PRINCIPAL_LEFTOVER=10    # Keep 10% of principal in reserve
```

This reserves units as an idle buffer. With 10 units and 10% leftover, 9 units trade and 1 stays idle.

Why keep a buffer?

1. **Slippage absorption**: if a buy costs slightly more than the unit size due to slippage, the extra USDC comes from the unallocated buffer
2. **Downside cushion**: if all active units get stopped out simultaneously, the buffer USDC remains in the wallet, preventing total capital deployment at unfavorable prices
3. **Recovery capacity**: after a drawdown, buffer units can be deployed into the new (lower) grid with fresh buy targets

Example: `PRINCIPAL=500` , `SPLIT_NUMBER=10` , `PRINCIPAL_LEFTOVER=10` :

- 9 units trade (\$450 deployed), 1 unit stays idle (\$50 buffer)
- SOL crashes and all 9 units get stopped out at -2%: you lose $9 \times \$1.00 = \9.00
- Your wallet still has \$50 (buffer) + \$441 (returned from stops) = **\$491 USDC**
- The grid resets around the new lower price, and now 9 units deploy fresh buy orders at the discounted level
- Compare to `PRINCIPAL_LEFTOVER=0` : all 10 units would have been stopped out, losing $10 \times \$1.00 = \10.00 , and no buffer remains

For live trading, **10% is a solid default**. Drop to 5% once you're comfortable with the bot's behavior. Only set 0% if you fully trust the parameters and understand you're maximizing exposure.

HAPPY_GAIN: taking profits off the table

```
HAPPY_GAIN=100    # Transfer profits when cumulative gains reach 100 USDC
```

When cumulative profit crosses this threshold, the bot sends exactly `HAPPY_GAIN` USDC to your `BENEFICIARY_WALLET` . Any excess carries over: if cumulative is \$103.50 and `HAPPY_GAIN=100` , it sends \$100.00 and keeps \$3.50 as the new running total. This is your **profit extraction mechanism**: gains move to a separate wallet where they are no longer at risk.

Size it relative to your capital and rhythm. With a \$250 unit (`PRINCIPAL=5000` , `SPLIT_NUMBER=20`) netting roughly \$4.25 per trade (Example C in Section 9), and ~2–3 winning trades per day, you accumulate ~\$10/day. `HAPPY_GAIN=100` extracts \$100 roughly every 10 days, a clean weekly-ish withdrawal without constant micro-transfers.

Frequent transfers (low `HAPPY_GAIN` , e.g. 5–10 USDC): you realize gains often, but each transfer costs a small SOL gas fee.

Infrequent transfers (high `HAPPY_GAIN` , e.g. 100 USDC): fewer transactions, meaningful withdrawal amounts, but more unrealized profit sitting in the operator wallet at risk in the meantime.

Rule of thumb: set `HAPPY_GAIN` to roughly 10–20× your expected daily profit. It should feel like a weekly paycheck, not a micro-drip.

6. Circuit Breaker: Stop-Loss Cascade Protection

During a sharp market drop, multiple stop-losses can fire within seconds. Without protection, the bot immediately re-enters at the new price, and if the market drops again, a second wave of stop-losses wipes more capital. The circuit breaker pauses new buys after a wave of stop-losses, giving the market time to stabilize.

How it works

The bot keeps a rolling list of recent stop-loss timestamps. Before opening any new buy, it checks:

1. Count stop-losses that fired within the last `CIRCUIT_BREAKER_WINDOW_MIN` minutes
2. If the count reaches `CIRCUIT_BREAKER_COUNT`, activate a cooldown
3. No new buys until the cooldown expires
4. Existing positions continue to sell at their normal targets

`CIRCUIT_BREAKER_WINDOW_MIN` is the bot's short-term memory for bad trades. Every stop-loss timestamp is recorded. Timestamps older than this window are forgotten. The bot only counts what happened *recently*: if two stop-losses are further apart than this window, the bot never connects them as part of the same downtrend and keeps buying.

Example: window too narrow (20 min): SL fires at 01:00. SL fires again at 01:22. The bot forgets the first one (>20 min old) before the second arrives. Count resets to 1. No circuit breaker. The bot keeps buying into the falling market.

Example: window wide enough (90 min): SL fires at 01:00. SL fires again at 01:22. Both are within the 90-min window. Count = 2. Circuit breaker fires. Buys paused.

Set the window wide enough to catch slow, grinding downtrends, not just flash crashes. A value of 60–90 minutes covers most real-world bleed patterns.

The four parameters

Parameter	Default	What it controls
<code>CIRCUIT_BREAKER_COUNT</code>	3	How many stop-losses within the window trigger the breaker
<code>CIRCUIT_BREAKER_WINDOW_MIN</code>	30	The bot's memory window: how far back it looks when counting recent stop-losses. Too short and it misses slow downtrends; too long and isolated bad trades from hours ago could unfairly halt a good session.
<code>CIRCUIT_BREAKER_COOLDOWN_MIN</code>	240	How long buys are paused after the breaker fires (4 hours default)
<code>CIRCUIT_BREAKER_MAX_TRIGGERS</code>	2	How many times the breaker can trigger before the bot goes sell-only for the rest of the day

All four are hot-reloadable. You can change them via the Settings panel without restarting, and `TAKE_PROFIT` also has a **change** button on its own card there, for when it is the only thing you want to touch. Note that a change applies to future buys only: positions already open keep the take-profit they were bought with, unless you retarget them explicitly (see "Changing take-profit with positions open").

Escalating response

If the breaker triggers multiple times in the same day, the response escalates:

Trigger # today	What happens
1st	4 hour cooldown, then resume
2nd	Sell-only mode for rest of day

With `CIRCUIT_BREAKER_MAX_TRIGGERS=2` (default), the bot gets one chance to recover. If the market crashes again after cooldown, it stops trying for the day.

Escalating cooldown formula: each successive trigger doubles the cooldown: $\text{base} \times 2^{(\text{trigger}-1)}$. So with a 240-min base: trigger 1 = 240 min, trigger 2 = 480 min, trigger 3 = 960 min, etc. With the default `MAX_TRIGGERS=2`, the 2nd trigger enters sell-only instead of applying a longer cooldown.

Walk-through examples

Flash crash (stops within seconds):

```

11:31  Crash: 10 stop-losses fire in under a minute
      3 in 30 min -> breaker ON (trigger 1 of 2)
      Bot pauses new buys until 15:31

14:26  Second crash happens
      Bot is still in cooldown, no units were bought
      No damage taken

15:31  Cooldown expires, bot resumes
      Rebuilds grid at post-crash price level

19:04  Recovery: bot catches profitable oscillation

```

Slow bleed (stops spread an hour apart, the dangerous pattern):

```

CIRCUIT_BREAKER_WINDOW_MIN=20 (too narrow):

01:00  SL #1 fires. Window memory: [01:00]. Count = 1.
01:22  SL #2 fires. 01:00 is now >20 min old, forgotten.
      Window memory: [01:22]. Count = 1. No breaker.
01:45  SL #3 fires. 01:22 is now >20 min old, forgotten.
      Window memory: [01:45]. Count = 1. No breaker.
      Bot keeps buying into a 3-hour downtrend. Full losses.

CIRCUIT_BREAKER_WINDOW_MIN=90 (wide enough):

01:00  SL #1 fires. Window memory: [01:00]. Count = 1.
01:22  SL #2 fires. Both within 90 min.
      Window memory: [01:00, 01:22]. Count = 2. BREAKER ON.
      Buys paused. Slow bleed stopped after just 2 losses.

```

Without the breaker, the bot would have redeployed at 11:31 and lost another wave of capital at 14:26.

The breaker does not cancel existing positions. Units already *HOLDING* continue to sell at their take-profit or stop-loss targets. The breaker only prevents new buys. The bot keeps running, the dashboard stays live, and Telegram alerts still work.

How it fits with other safeguards

Feature	Role	When it acts
STOP_LOSS	Cuts one bad position	Per unit, immediately
Circuit breaker	Pauses after a wave of stops	After N stops in T minutes
MAX_DAILY_LOSS	Hard daily limit	After cumulative losses exceed threshold
Wind-down	Graceful exit	Manual, or automatic on 2nd breaker trigger

They form a chain: STOP_LOSS, then circuit breaker, then MAX_DAILY_LOSS, then wind-down. Each catches what the previous one missed.

7. Capital Sizing and Unit Allocation

```
PRINCIPAL=500          # Total USDC budget allocated to trading
SPLIT_NUMBER=10        # Number of grid slots (trading units)
PRINCIPAL_LEFTOVER=10  # % of principal held in reserve as safety buffer
```

- **PRINCIPAL:** the total USDC the bot can deploy. Unit size is always `PRINCIPAL / SPLIT_NUMBER`.
- **SPLIT_NUMBER:** how many grid levels to create. More units means a denser grid with more buy opportunities at different price levels.
- **PRINCIPAL_LEFTOVER:** percentage kept idle as a buffer. At 10% with 10 units, 9 units trade actively and 1 stays idle as a cushion.

PRINCIPAL and SPLIT_NUMBER have no default. Both lines must be present in `config.env` with a value. If either is missing, commented out, or empty, the bot refuses to start and names the key. A fallback for these two would be a 100 USDC, 10 unit grid, which is nobody's actual capital. A bot that quietly deploys a tenth of your wallet because a line was deleted is a worse outcome than one that will not start.

Every other parameter falls back to a default, but set `TAKE_PROFIT` yourself. Its default is 0.7, which is also the value `config.env.example` ships, and the bot's settings checks refuse 0.7 in both modes (it will still start and trade with it): it is at or below the 1.60% live cost floor at `SLIPPAGE_BPS=50`, and below the 1.00% paper floor at the default costs. Set it above 1.60% for live trading or above 1.00% for paper trading. Until you do, every settings change you make, from the dashboard or by editing `config.env`, is refused unless it also raises `TAKE_PROFIT`. The short examples in this guide that set only one or two keys assume you have already done this.

How much capital?

Magellan works at any scale, but there's a practical minimum:

PRINCIPAL	SPLIT_NUMBER	Unit Size	Viability
\$50	5	\$10	Minimum viable - small trades, limited grid depth
\$100	10	\$10	Good starting point for paper testing
\$500	10	\$50	Solid for live trading - meaningful per-trade profit
\$1,000	15	~\$67	Good density - 15-level grid catches more oscillations
\$5,000	20	\$250	Deep grid with significant per-trade returns

How many units?

More units = denser grid = more buy levels. But there are diminishing returns:

- **5 units:** Sparse grid. Okay for testing with small capital.
- **10 units:** The sweet spot for most traders. Good grid density without overcomplicating things.
- **15–20 units:** Dense grid, catches more price levels. Best with \$1,000+ to keep unit sizes meaningful.
- **25+ units:** Very dense. Risk of many units buying at nearly the same price during a dip. Only if you really know what you're doing.

The MIN_TRADE_USDC guard

```
MIN_TRADE_USDC=1    # Skip trades below 1 USDC
```

`MIN_TRADE_USDC` is a safety guard, not a limiter on your regular trade size. Your actual trade size is always `PRINCIPAL / SPLIT_NUMBER` (the unit size). This parameter only triggers when a unit's available balance is unexpectedly small, due to rounding, leftover dust from a previous swap, or partial fills.

Example: unit size is \$50, but after rounding a unit has only \$0.08 USDC left. Without this guard, the bot would try to swap \$0.08, paying gas on a trade that returns almost nothing. With

`MIN_TRADE_USDC=1`, the bot skips it entirely.

In normal operation this guard never triggers. It is a last-resort defence against dust trades. The default value of `1` USDC is suitable for most setups.

8. Reading the Market: When to Use Which Settings

The presets below assume 9 active units (`SPLIT_NUMBER=10` , `PRINCIPAL_LEFTOVER=10`); with more active units, `GRID_SPREAD` has to shrink so that `GRID_SPREAD × active_units` stays at or below `STOP_LOSS` .

Choppy / sideways market (best case)

What is this? Price oscillates within a narrow range with no sustained direction. SOL moves up \$3, back down \$3, up again, repeatedly. There is no trend. This is Magellan's natural habitat: the grid fills on dips and sells on small recoveries, cycling through positions all day without needing the market to go anywhere.

SOL bouncing between \$147–\$153 with no clear trend.

```
TAKE_PROFIT=1.8
STOP_LOSS=2
GRID_SPREAD=0.2
GRID_RESET_THRESHOLD=2
```

Why: a take-profit only 0.2% above the 1.60% live cost floor catches the most oscillations while still clearing it. Grid spread is narrow because price isn't moving far. Stop-loss is wide enough to avoid false triggers.

Expected behavior: many units cycle through buy→sell within hours. High trade volume, consistent small profits.

Play-by-play (\$500 principal, 10 units, \$50/unit):

```
08:00 SOL $150.00 - Grid placed. Unit 1 target: $149.70, Unit 2: $149.40...
09:10 SOL $149.35 - Units 1-2 buy
11:30 SOL $152.40 - Units 1-2 sell (+1.8% each) → +$1.80 profit
13:00 SOL $148.75 - Units 3-4 buy (Units 1-2 recycled to the bottom of the ladder)
15:30 SOL $151.80 - Units 3-4 sell → +$1.80 profit
17:00 SOL $147.85 - Units 5-7 buy (deeper dip)
20:30 SOL $151.20 - Units 5-7 sell → +$2.70 profit
      Daily total: 7 round-trips, ~$6.30 gross profit, zero stop-losses
```

Volatile market (moderate swings)

What is this? Price makes large intraday moves in both directions: drops of 3–5% followed by recoveries of similar size. The market has energy and range but no clear direction. More opportunity per trade, but also more risk, a 5% drop can trigger stop-losses before the recovery comes.

SOL swinging 3–8% intraday.

```
TAKE_PROFIT=2.0
STOP_LOSS=3
GRID_SPREAD=0.3
GRID_RESET_THRESHOLD=3
```

Why: wider take-profit and grid spread match the larger swings. Higher stop-loss gives positions room to survive temporary dips before recovering.

Expected behavior: fewer but larger profits. Some units may hold for hours waiting for the recovery swing. Occasional stop-losses during sharp drops, but winning trades more than compensate.

Trending up

What is this? Price climbs steadily over hours or days with only minor pullbacks. SOL goes from \$140 to \$160 with brief dips along the way. Magellan can still profit on the pullbacks within the trend, but it needs to reset its grid frequently to stay relevant as price moves higher.

SOL climbing steadily from \$140 to \$160 over a few days.

```
TAKE_PROFIT=2.0
STOP_LOSS=2
GRID_SPREAD=0.2
GRID_RESET_THRESHOLD=1.5
```

Why: lower grid reset threshold keeps the grid chasing the price upward. Units buy on pullbacks and sell on the next leg up. Works well as long as the trend has regular pullbacks.

Expected behavior: grid resets frequently as price climbs. Each reset anchors new buy targets just below the new price. Units cycle on pullback oscillations within the trend.

Warning: a clean uptrend with no pullbacks means no buy triggers. The bot sits idle waiting for a dip. That's fine. It means you're not buying into an overextended move.

Trending down

What is this? Price falls steadily with brief recoveries that don't hold. This is the hardest environment for grid trading: the bot buys what looks like a dip, but price keeps falling, turning each buy into a stop-loss. Capital preservation is the priority here, not profit.

SOL falling steadily from \$160 to \$140.

This is the **hardest market** for grid trading. Units buy on what looks like a dip, but the price keeps falling, triggering stop-losses.

```
TAKE_PROFIT=2.0
STOP_LOSS=1.5
GRID_SPREAD=0.15
GRID_RESET_THRESHOLD=3
MAX_DAILY_LOSS=3
GLOBAL_SL_COOLDOWN_MIN=120
CIRCUIT_BREAKER_COUNT=2
CIRCUIT_BREAKER_WINDOW_MIN=90
CIRCUIT_BREAKER_COOLDOWN_MIN=480
CIRCUIT_BREAKER_MAX_TRIGGERS=1
```

Why: tighter stop-loss limits damage per position. Grid spread is narrow because a 1.5% stop-loss leaves room for a grid only 1.5% deep on 9 units ($0.15\% \times 9 = 1.35\%$), and the bot refuses a deeper one, so the protection in this market comes from the cooldown and circuit breaker settings below rather than from spreading units apart. Low daily loss cap halts trading early. High reset threshold prevents the grid from chasing the price down.

GLOBAL_SL_COOLDOWN_MIN=120 : after any stop-loss, ALL buys pause for 2 hours. If the market is still falling when the pause expires and a second stop-loss fires, the timer resets for another 2 hours. This is the most direct protection against buying into a sustained downtrend.

CIRCUIT_BREAKER_WINDOW_MIN=90 : widens the bot's memory window to 90 minutes so it catches slow bleeds: two stop-losses that are 30, 60, or even 80 minutes apart still count as a pattern. The default 30 minutes only catches flash crashes, not hour-long slides.

CIRCUIT_BREAKER_COUNT=2 + **CIRCUIT_BREAKER_MAX_TRIGGERS=1** : two stop-losses within 90 minutes triggers the breaker immediately, and the first trigger goes straight to sell-only for the rest of the day (no second chance). In a genuine downtrend, giving the bot a second chance usually means a second wave of losses.

Expected behavior: some stop-losses triggered. Global SL cooldown and circuit breaker cut buying long before the daily loss limit is reached. The bot protects capital and waits for conditions to improve.

Play-by-play (\$500 principal, 10 units, \$50/unit):

```

08:00 SOL $160 - Grid placed. Units 1-9 waiting at $159.76 to $157.84
09:30 SOL $158.60 - Units 1-5 buy (targets hit)
10:15 SOL $157.80 - Units 6-9 buy, Units 1-5 still holding (waiting for recovery)
11:00 SOL $156.20 - Units 1-5 hit stop-loss at -1.5% → sell → -$3.75 total loss
      → 5 SLs in 90 min ≥ CB_COUNT=2 → circuit breaker fires
      → MAX_TRIGGERS=1 → sell-only for rest of day
      GLOBAL_SL_COOLDOWN=120 min also blocks any new buys until 13:00
11:00 Units 6-9 still holding, waiting for +2.0% recovery
12:30 SOL $155.40 - Units 6-9 hit stop-loss → sell → -$3.00 total loss
12:30 Daily P&L: -$6.75 → well under $15 limit (3% of $500)
      Bot in sell-only mode. No grid reset, no new buys.
16:00 SOL $140 - Bot idle since 12:30. Zero additional losses.
      Circuit breaker saved ~$8 by stopping buys after the first wave

```

Low volatility / dead market

SOL stuck at \$150.00 ± \$0.20 for days.

Grid trading doesn't work here. There's not enough movement to trigger buys, or if units buy, the price doesn't move enough to hit take-profit.

Best approach: either leave the bot running with Level 1 (Conservative) settings and accept low activity, or pause it and wait for volatility to return. Don't lower `TAKE_PROFIT` below your cost threshold just to get trades. You'll lose on every cycle.

9. Worked Examples

Example A: Conservative live setup (\$500)

You want to trade real capital with minimal risk.

```
MODE=live
PRINCIPAL=500
SPLIT_NUMBER=10
PRINCIPAL_LEFTOVER=10
TAKE_PROFIT=2.0
STOP_LOSS=3
GRID_SPREAD=0.3
GRID_RESET_THRESHOLD=3
MAX_DAILY_LOSS=3
HAPPY_GAIN=5
SLIPPAGE_BPS=50
```

Breakdown:

- Unit size: $\$500 / 10 = \text{\$50 per unit}$
- Active units: 9 (1 buffer)
- Grid depth: $0.3\% \times 9 = \textbf{2.7\% below market}$ (deepest unit), inside the 3% stop-loss
- Profit per winning trade: $\$50 \times 2.0\% = \text{\$1.00 gross}$, minus $\sim \$0.15$ fees $\approx \text{\$0.85 net}$
- Daily loss cap: $\$500 \times 3\% = \text{\$15 max daily loss}$
- Stop-loss damage: $\$50 \times 3\% = \text{\$1.50 per losing trade}$ (plus $\sim \$0.15$ fees = $\$1.65$ total)
- Wins needed to cover 1 loss: $\$1.65 / \$0.85 \approx \textbf{2 wins}$

With SOL oscillating 3–5% intraday, you might see 1–4 completed trades per day, mostly winners. A realistic daily return: **\\$1–3** (0.2–0.6% of principal).

Example B: Active paper testing (\$100)

You want to see lots of trades and understand the bot's behavior.

```

MODE=paper
PRINCIPAL=100
SPLIT_NUMBER=10
PRINCIPAL_LEFTOVER=10
TAKE_PROFIT=0.3
STOP_LOSS=1.5
GRID_SPREAD=0.15
GRID_RESET_THRESHOLD=1.5
MAX_DAILY_LOSS=7
SLIPPAGE_BPS=10
PAPER_DEX_FEE=0.0005

```

Breakdown:

- Unit size: **\$10 per unit**
- Active units: 9 (1 buffer from 10% leftover)
- Grid depth: $0.15\% \times 9 = \mathbf{1.35\% \text{ below market}}$ (deepest unit), inside the 1.5% stop-loss
- Paper round-trip cost: $\sim 0.2\%$
- Net profit per trade: $0.3\% - 0.2\% = \mathbf{\sim 0.1\% \approx \$0.01 \text{ per trade}}$
- Tight margins, but you'll generate many trades for analysis

Use the dashboard's History tab to study patterns: when do stop-losses cluster? What time of day generates the most wins? How often does the grid reset?

Example C: Scaling up (\$5,000)

You've validated the strategy in paper mode and small live, now deploying serious capital.

```

MODE=live
PRINCIPAL=5000
SPLIT_NUMBER=20
PRINCIPAL_LEFTOVER=10
TAKE_PROFIT=2.0
STOP_LOSS=2
GRID_SPREAD=0.1
GRID_RESET_THRESHOLD=2
MAX_DAILY_LOSS=3
HAPPY_GAIN=20
SLIPPAGE_BPS=50

```

Breakdown:

- Unit size: $\$5,000 / 20 = \textbf{\$250 per unit}$
- Active units: 18 (2 buffer)
- Grid depth: $0.1\% \times 18 = \textbf{1.8\% below market}$ (deepest unit), inside the 2% stop-loss
- Profit per winning trade: $\$250 \times 2.0\% = \textbf{\$5.00 gross}$, minus $\sim \$0.75$ fees $\approx \textbf{\$4.25 net}$
- Daily loss cap: $\$5,000 \times 3\% = \textbf{\$150 max daily loss}$
- Stop-loss damage: $\$250 \times 2\% = \textbf{\$5.00 per losing trade}$
- Wins needed to cover 1 loss: $\$5.00 / \$4.25 \approx \textbf{1.2 wins}$

With 18 active units packed into a 1.8% grid to stay inside the 2% stop-loss, one dip can fill most of them at once. Realistic daily return: **\$10–30** (0.2–0.6% of principal). Profits auto-transfer every time \$20 accumulates.

10. The Paper-to-Live Pipeline

Don't go straight from zero to live trading. Follow this pipeline:

Stage 1: Paper with aggressive settings (1–2 days)

```
MODE=paper  
TAKE_PROFIT=1.2  
GRID_SPREAD=0.2
```

Goal: generate lots of trades to understand how the grid behaves. Watch the dashboard. Look at:

- How many trades per hour?
- When do stop-losses trigger?
- How does the grid reset look?
- Is the daily P&L positive?

Stage 2: Paper with your intended live settings (3–5 days)

```
MODE=paper  
TAKE_PROFIT=2.0  
GRID_SPREAD=0.2
```

Goal: validate that your actual configuration is profitable. Collect enough data for statistical significance (at least 20–30 completed trades). Check the History tab:

- Is win rate above 75%?
- Is cumulative P&L trending up?
- Are stop-losses rare (<10% of trades)?

Stage 3: Live with small capital (1–2 weeks)

```
MODE=live  
PRINCIPAL=50  
# keep the TAKE_PROFIT and GRID_SPREAD you validated in Stage 2
```

Goal: verify real execution. Jupiter swaps, actual slippage, real gas costs. Compare live P&L to paper P&L. Live will typically be slightly worse due to real-world execution costs.

Stage 4: Scale up

Once live results match or are close to paper results over 1–2 weeks, gradually increase **PRINCIPAL**.
Don't jump from \$50 to \$5,000. Go \$50 → \$200 → \$500 → \$1,000 → target.

11. Common Mistakes

Setting TAKE_PROFIT too low

If `TAKE_PROFIT` doesn't exceed your round-trip trading costs, every "winning" trade actually loses money. The bot enforces a minimum on changes: the Settings dialog, the dashboard server and a `config.env` reload all refuse a `TAKE_PROFIT` at or below its worst-case round-trip estimate. In live mode that is **1.60% at the default** `SLIPPAGE_BPS=50`, however cheaply your fills actually execute; in paper mode the floor follows your simulated costs (1.00% at the defaults). A restart does not re-check, so a value already in `config.env` when the bot starts is traded as is (paper mode only prints a console warning).

Example: you set `TAKE_PROFIT=0.3` with a \$50 unit. The bot buys at \$149.25, sells at \$149.70 for a gross profit of \$0.15. But Jupiter fees + slippage cost \$0.15 on the buy and \$0.10 on the sell = \$0.25 total fees. Net result: **-\$0.10 loss on a "winning" trade**. Repeat 20 times a day and you lose \$2/day while the dashboard shows positive trades.

Ignoring the stop-loss / take-profit ratio

A 0.5% take-profit with a 3% stop-loss sounds like 6:1, but costs make it far worse. After ~0.3% round-trip costs, your net win is ~0.2% while your actual loss is ~3.15% (stop-loss + sell fees). That's **~16 wins needed per loss**. If your win rate drops below 94%, you bleed capital. Keep the ratio manageable. Ideally `STOP_LOSS` should be no more than 3–4× your `TAKE_PROFIT`.

Example: with \$50 units, `TAKE_PROFIT=0.5` (net ~\$0.10/win after fees) and `STOP_LOSS=3` (loss = \$1.58 with fees). You need 16 winning trades to recover from a single stop-loss. If you get 2 stop-losses in a row, that's 32 wins needed just to break even.

Grid spread too narrow with many units

`GRID_SPREAD=0.1` with 20 units means all 20 units buy within a 2% range below market. If SOL drops 2% and then drops another 3%, you have 20 units all underwater together. Wider spread = better risk distribution, up to the depth rule the bot enforces (see `GRID_SPREAD` in Section 4).

Example: SOL at \$150, `GRID_SPREAD=0.1`, 20 units. All units have buy targets between \$149.85 and \$147.00. SOL drops to \$147 → all 20 units buy. SOL continues to \$143 → all 20 units hit stop-loss simultaneously. Loss: $20 \times \$1.50 = \mathbf{\$30.00 \text{ in one move}}$. A wider spread helps only up to that limit: at this 3% stop-loss, 20 active units can be spread no wider than 0.15% (targets spanning \$149.78 to \$145.50). That leaves 7 of them unfilled at \$147, but a fall on to \$143 still reaches every target.

Chasing losses by widening daily loss limit

When `MAX_DAILY_LOSS` triggers, it's doing its job. Don't increase it just because you hit it once. A day where you hit the loss limit is a day where the market conditions were bad for grid trading. The bot stopped you from losing more. That's a win.

Example: your bot hits `MAX_DAILY_LOSS=3` (\$15 loss on \$500 principal) during a SOL selloff. You think "the selloff is almost over" and increase to `MAX_DAILY_LOSS=10`. SOL keeps dropping, the bot buys into every dead-cat bounce and stops out each time. End of day: **-\$42 loss** instead of -\$15. The original limit would have saved you \$27.

Running on a flaky RPC

Jupiter swaps involve multiple API calls (order → sign → execute). If your RPC is slow or unreliable, transactions may timeout and retry, wasting gas and creating edge cases. Use a reliable RPC (Helius recommended) and enable `FALLBACK_ENABLED=Y` with a `FALLBACK_RPC_URL` pointing to a second provider. During network congestion, if you see repeated `TX_FAILED` events in the logs, enable `PRIORITY_FEE=Y` with a higher `PRIORITY_FEE_LAMPORTS` value (e.g., 100000 for moderate congestion).

Never checking the dashboard

The dashboard exists for a reason. Check it daily. Look at:

- **Win rate:** if it drops below 70%, your parameters may need adjusting
- **Trade feed:** are stop-losses clustering at a particular time?
- **Grid resets:** too many resets mean your `GRID_RESET_THRESHOLD` is too low
- **Cumulative P&L:** the only number that matters long-term

Not exporting trade history

Use the **Export CSV** button in the History tab to download the day on screen, or **Advanced Export** beside it for a date range of up to 366 days in one file: every trade from both engines, in the same columns, oldest first. If any day in the range cannot be read, it exports nothing rather than a file with a day missing. Open them in a spreadsheet and analyze: which times of day are most profitable? Do certain units perform better? Is there a pattern to stop-loss events? Data-driven tuning beats guesswork.

12. Advanced Tactics

Adjusting parameters by time of day

SOL volatility isn't uniform. US market hours (14:30–21:00 UTC) and Asian market hours (01:00–08:00 UTC) tend to be more volatile. You can:

1. Run with tighter settings (lower TAKE_PROFIT, narrower GRID_SPREAD) during high-volatility hours
2. Switch to wider settings during quiet hours

Hot-reloadable parameters (`TAKE_PROFIT` , `STOP_LOSS` , `GRID_SPREAD` , `GRID_RESET_THRESHOLD` , `MAX_DAILY_LOSS` , and others) can be changed directly from the **Settings panel** in the dashboard. Changes take effect on the next poll cycle, no restart required. For time-based strategies, you can script config swaps against `config.env` using a cron job, but manual adjustments via the dashboard are sufficient for most use cases.

One thing to know: a hot-reload applies to *future buys only*. Positions you already hold keep the take-profit they were bought with. See "Changing take-profit with positions open" below.

Changing take-profit with positions open

Each unit stores its sell target at the moment it buys. If you bought at 4% and then lower `TAKE_PROFIT` to 3%, that open position still sells at 4%. This is deliberate: a settings edit should not silently move the exit price out from under money you already have in the market.

When you want them moved anyway, change `TAKE_PROFIT` from the **Settings** panel. Two routes get you there and both behave identically: the **change** button at the top of the panel, which opens every hot-reloadable parameter at once, or the small **change** button in the corner of the `TAKE_PROFIT` card itself, which opens the same dialog with just that one field, already focused so you can type straight over it and press Enter. Use the second when take-profit is the only thing you are touching, which is most of the time.

Either way the confirmation dialog lists every open position and what the change would do to it:

- **Entry, current target, new target:** what it bought at, what it sells at today, what it would sell at.
- **To new TP:** how far the price still has to travel to reach the new target, in percent and in USDC. This figure is gross: trading fees come out of it.
- **Status:** *Safe*; ⚡ *Sells within moments* when the position is within 0.1% of its new target; or ⚠ *Sells immediately* when the new target is already below the current price. The warning means the sale happens right away, not that it loses money: a position past its take-profit is in profit.

Every position that is not already past its new target is ticked by default, including the *Sells within moments* ones: closing is what you asked for. Anything flagged *Sells immediately* is left unticked, because moving it would fire a sale on the next tick. You can still tick it on purpose if that sale is what you want: nothing is hidden from you, it just isn't the default.

Positions you leave unticked keep their existing target and carry on untouched. `STOP_LOSS` is never retargeted: an immediate stop-out realizes a real loss, which is a very different thing from an early profit.

Two practical notes. The bot has to be running: the retarget is applied on its next tick, and the dashboard will refuse to queue one if the bot looks stopped. And the price is re-checked at that moment, not when you clicked: if the market moves in the seconds in between, a position that looked safe can still be skipped, and the log will say so.

Example setup with two config files:

```
# config.env.active - for volatile hours (US/Asia market overlap)
TAKE_PROFIT=1.8
GRID_SPREAD=0.15

# config.env.quiet - for low-volatility hours (overnight US)
TAKE_PROFIT=2.2
GRID_SPREAD=0.2
```

Each file must be a **complete copy** of `config.env` (`PRINCIPAL` , `SPLIT_NUMBER` , `MODE` , wallet keys and all) with only these lines changed, because `cp` replaces the whole file and the bot refuses to start without its sizing lines (and, in live mode, its RPC and wallet lines).

```
# Cron jobs (add via `crontab -e`). Cron does not run in the Magellan folder, so use
full paths.
# No restart: the bot watches config.env and hot-reloads these fields.
# Switch to active at 14:30 UTC (US market open)
30 14 * * 1-5 cp /var/www/magellan/current/config.env.active /var/www/magellan/
current/config.env
# Switch to quiet at 21:00 UTC (US market close)
0 21 * * 1-5 cp /var/www/magellan/current/config.env.quiet /var/www/magellan/current/
config.env
```

Keep `PRINCIPAL` and `SPLIT_NUMBER` in both copies identical to what the bot is running, or a copy made before a capital change quietly puts the old capital back into `config.env` . And each copy has to pass the reload's cross-field checks, or the bot refuses the reload: its `TAKE_PROFIT` has to clear the round-trip cost floor (see "The dashboard checks this for you"), and its `GRID_SPREAD` times your active units must stay at or below `STOP_LOSS` (on 9 active units with `STOP_LOSS=2` , a copy's spread can be at most $2 / 9 \approx 0.22$, so the active copy's 0.15 and the quiet copy's 0.2 both fit, 1.35% and 1.8% deep).

Using GRID_SPREAD=0 for conviction plays

Setting `GRID_SPREAD=0` makes all units buy at the same price. This is essentially an **all-in** at a single level, useful if you're highly confident SOL will bounce from a specific support level.

```
GRID_SPREAD=0
TAKE_PROFIT=2.0
```

All 9 active units buy together (at `SPLIT_NUMBER=10`, `PRINCIPAL_LEFTOVER=10`), and a 2% bounce gives you 9× the single-unit profit. But if the bounce doesn't come, all 9 take the stop-loss together. High reward, high risk.

Stacking HAPPY_GAIN for compound growth

Set `HAPPY_GAIN` high enough that profits stay in the operator wallet and effectively increase your trading capital between transfers. The bot trades with `PRINCIPAL`, but if your wallet has more USDC than `PRINCIPAL`, the excess acts as additional buffer.

Example: `PRINCIPAL=500`, `HAPPY_GAIN=50`. The bot accumulates up to \$50 in profit before transferring. That extra \$50 in the wallet absorbs slippage and provides a larger cushion against stop-losses.

Monitoring with the dashboard CSV export

Export a date range with **Advanced Export** and open it in a spreadsheet. Create charts of:

- **Win rate over time:** is it trending up or down?
- **Average profit per trade:** are costs eating more of your margin?
- **Stop-loss frequency:** are losses clustered or evenly distributed?
- **Trades per hour heatmap:** which hours generate the most (and most profitable) activity?

This analysis tells you when your parameters need updating better than any single metric.

Fallback RPC for reliability

For serious live deployments, always enable fallback:

```
FALLBACK_ENABLED=Y
FALLBACK_RPC_URL=https://solana-mainnet.g.alchemy.com/v2/YOUR_KEY
```

If your primary RPC goes down, the bot automatically switches. Downtime = missed trades = missed profits.

13. Trading Tokenized Stocks

Magellan can trade a second market: **tokenized equities on Solana**, tokens that track a real share price. It runs as a **completely separate engine** from the SOL/USDC bot, with its own wallet, its own capital, its own configuration file, its own state and logs, and its own process. Nothing in this chapter changes anything about the grid you have been reading about so far.

That separation is the point. The two engines cannot spend each other's money, cannot halt each other, and cannot corrupt each other's records. If you never turn the stocks engine on, Magellan behaves exactly as it always has.

The one idea to take away

Take profit from the token's price move. Use the peg only as a guard.

A tokenized stock has two prices: the price the **token** trades at on Solana, and the **mark**, which is what the underlying share is worth. They normally track each other closely, but they drift.

The temptation is to trade the drift, buying whenever the token looks cheap against the mark. The engine deliberately does not do that. It runs the same grid-and-take-profit strategy you already know, on the token's own price, and uses the mark for exactly two jobs:

- **blocking a buy** that is too expensive relative to the real share
- **blocking a sell** that would hand back your gain because the peg is distorted at that moment

The discount is never the profit target. It is a safety rail.

The allowed band

The gap between the two prices is expressed as a discount:

$$\text{discount \%} = (\text{mark price} - \text{token price}) / \text{mark price} \times 100$$

A **positive** discount means the token is cheaper than the share. A **negative** discount means it is more expensive: a premium.

Two settings define the band the engine will act inside:

Setting	Default	Meaning
<code>DISCOUNT_MIN_PCT</code>	-0.5	how much of a premium is tolerable
<code>DISCOUNT_MAX_PCT</code>	+2.0	how large a discount still looks like a working peg

A discount **wider** than the maximum is not a bargain, it is a warning. A peg that far out of line usually means something is wrong with the market for that token, and the engine treats it as a reason to wait rather than an opportunity.

How a buy is decided

A unit buys only when **all** of these hold:

1. the engine is allowed to trade this name: it is in the live top `STOCK_COUNT` , **or** the engine already holds a position in it and it still passes the screening filters (see "Committing to a stock" below)
2. the discount sits inside the band
3. liquidity and 24h volume still pass the screening filters
4. the token's price has fallen to that unit's grid target

Anything else is refused, and the reason is written to the log. If you ever suspect the engine of being too cautious, those `BUY_SKIPPED` lines are the ones to read: each names the condition that blocked it.

How a sell is decided, and the three-way check

A sell is triggered the ordinary way, by the token's own price reaching your take-profit or your stop-loss. Before it sends, the engine checks the mark one more time:

Discount at the trigger	What happens
inside the band	sell normally
below <code>DISCOUNT_MIN_PCT</code> (the token is rich)	sell anyway , log <code>EXIT_WHILE_PREMIUM</code> , and never reopen that name
above <code>DISCOUNT_MAX_PCT</code> (the peg is broken)	do not sell ; wait for the peg to come back, unless a stop-loss has been waiting 30 minutes
the mark cannot be read at all	do not sell , same 30-minute limit on a stop-loss

The middle row is the one worth understanding. Selling into a premium is fine: you are being paid more than the share is worth. But a name that behaves that way once has shown you something about itself, so the engine refuses to buy it again. That is permanent, and it survives a restart.

The third row is the opposite case. A sell into a badly broken peg gives away your edge, so the take-profit stays armed and the sale waits. It is not cancelled.

The last two rows are a deliberate trade-off, and the limit on them is worth knowing before it matters. The mark comes from the same price feed as everything else, so if that feed is unreachable the engine cannot check the peg, and a check it cannot run is treated as a check that

failed. It holds the position rather than selling blind. That case is logged distinctly (`SELL_DEFERRED` with `reason: mark_unavailable`) so it does not look like an ordinary hold, and `checkScreener` reports mark availability for every ranked name so you can see it coming.

A take-profit can wait indefinitely. A stop-loss waits 30 minutes and then goes out anyway. The two are not the same trade, and treating them the same was a real gap. Deferring a take-profit costs nothing: the position is temporarily underpriced, so waiting means selling it for more. Deferring a stop-loss removes your protection at the moment it exists for, and the usual objection does not apply, because a stop is already selling low by definition. "The token is far cheaper than the real share" is also exactly what a depeg looks like from the inside, so an unbounded wait meant a broken token could ride a position down with nothing but a log line per tick.

Thirty minutes is long enough for a thin-book air pocket or a brief feed outage to pass without dumping a position into it, and short enough that a genuine depeg cannot run unattended. The trigger is re-checked against the live price every tick, so a position that recovers above its stop stops triggering and the clock is thrown away: the window only ever runs while the stop is still genuinely breached. When it does fire, the trade is still a `STOP_LOSS`, and the Telegram alert adds a line saying the peg was distorted and the sale was held back and then sent anyway. The log row carries `guardOverridden` too. The dashboard shows it as an ordinary stop-loss, because the feed translator publishes a fixed set of fields and this is not one of them. The clock is stored on the unit, so it survives a restart.

Capital, split two ways

Where the SOL/USDC bot divides capital by `SPLIT_NUMBER`, the stocks engine divides it twice:

```
unit size = PRINCIPAL x (1 - PRINCIPAL_LEFTOVER / 100) / (SPLIT_NUMBER x STOCK_COUNT)
```

With `PRINCIPAL=1000`, `SPLIT_NUMBER=5` and `STOCK_COUNT=3`, that is **66.67 USDC per unit**: five rungs on each of three ladders. Forgetting the `STOCK_COUNT` factor would commit three times the capital you actually have, which is why the engine refuses to start on a configuration that does not add up.

PRINCIPAL is the TOTAL across all names, not the amount per name. This is the one thing to get right before you change `STOCK_COUNT`. Raising `STOCK_COUNT` on its own does not commit more money: it divides the same money into smaller units. Going from one name to two at `PRINCIPAL=100` and `SPLIT_NUMBER=5` takes your unit from 20 USDC to 10, and the engine still deploys 100 in total.

To size it the way you are actually thinking about it, work backwards:

```
PRINCIPAL = number of names x SPLIT_NUMBER x the unit size you want
```

Two names of five units at 20 USDC each is $2 \times 5 \times 20 = 200$. Set `PRINCIPAL=200`, and make sure the wallet actually holds that much: the engine sizes orders from this number, not from your balance.

Committing to a stock

`STOCK_COUNT` is not "how many names the engine watches". It is **how many names it commits PRINCIPAL and SPLIT_NUMBER to at once**, and a name stays committed until that capital comes back.

A slot is held by money the engine has actually **spent**: a unit that is holding a position, or one with a buy or sell in flight. A unit that is only *waiting* at a grid target has spent nothing, and that money is still sitting in your wallet, so it holds no slot.

Two consequences, and they are the whole behaviour:

- **A stock you own keeps trading its whole ladder even when it drops out of the top ranking.** Ranking is how a name gets *chosen*; it is not a condition the name has to keep meeting to be allowed to finish. If you hold two of five units of SPYx and SPYx slips to second place, the other three units go on filling and recycling exactly as before. The engine only stops buying a name that genuinely **fails** the screen: peg outside the band, liquidity gone, volume gone. Such a name winds down, selling what it holds and buying nothing more.
- **A stock the engine has not actually bought can be swapped for a better one at any time.** If every unit of a grid is idle or merely waiting, the engine drops that name the moment something better ranks above it, and opens the new one instead.

So at `STOCK_COUNT=1` the engine picks a name, commits to it, and stays with it until it is flat. It does not chase a better name while your money is in the current one.

How often the tradable list is rebuilt

`RANKING_REFRESH_TICKS` (default `30`) sets how many ticks pass between rebuilds of the list of names the engine may trade.

It is counted in ticks, and one tick is the `POLL_INTERVAL_MS` in `config_stocks.env`, which is not the same number as the one in `config.env`. The SOL bot ships `10000`, this engine ships `2000`. At `2000`, the default of 30 ticks is **60 seconds**. Convert against the wrong file and you will be out by a factor of five.

Raising it costs Jupiter calls and nothing else that matters: one rebuild is five API calls, and the figures that decide a trade (price, mark, discount, liquidity) are re-read on **every** tick regardless. Only the 24h volume and the verified flag come from the rebuilt list, and neither moves quickly. Raising it also settles the grids down if you see the engine swapping names back and forth while it holds nothing.

Turning it on, and the two switches

The engine ships **off**, twice over, and both switches must be on before it can place an order:

- `MODE=live` in `config_stocks.env` (it ships as `paper`)
- `TRADE_STOCK=YES` in `config_stocks.env` (it ships as `NO`)

There is also an optional master switch: `TRADE_STOCK` in the SOL bot's own `config.env` , which the stocks engine **reads and never writes**. Leave it absent and the engine defers to its own setting.

Run it in paper mode first, for at least a few days. Paper mode uses real prices and real screening and simulates only the fills, so what you see is what the strategy would have done.

Watching it

Everything appears in the same dashboard:

- **Stock Operator Wallet** on the Dashboard tab: USDC in positions, USDC waiting, and the free surplus, with a Withdraw button that sends to the stocks beneficiary wallet
- **Stock Grids** in Live Status: how many names are being traded and how many units hold a position
- the **Live Trade Feed** on the **Trading** tab, and the **History** tab, where stocks trades appear beside SOL ones with the instrument named in its own column as a pair: `SPYx/USDC` beside `SOL/USDC` . The CSV export carries the same column
- the History **chart**, which plots SOL/USDC. On a day with stocks activity a selector above it offers **SOL/USDC** or **Stocks**, and Stocks shows no chart: names trading from under \$100 to several hundred dollars share no useful price axis, so their trades are read from Trade History
- **Trade History** hides grid-reset rows by default, because a price bouncing around the reset threshold can log one every few seconds. **View All** beside the heading shows them again, and Export CSV always includes them
- **Settings > Stocks**, which is the engine's own pane: the Top 10 screener, every parameter as a card with an info icon explaining it, both wallet addresses with a copy button and a Solscan link, and a **Configuration History** timeline showing what changed between engine starts

Changing the stocks engine's capital. Edit `config_stocks.env` and run `pm2 restart magellan-stocks` . On the way up, the engine reshapes its existing ladders to the new `SPLIT_NUMBER` and restamps every free unit to the new size, so the number of units and the size of each one always describe the same grid. Units that are holding something are never touched: if a shrink cannot finish because too many are busy, the ones left over are never raised to the new size (a smaller one, after a `PRINCIPAL` cut, is still applied) and are removed at the next restart once they free up. The log line says exactly what it did. Your statistics are unaffected either way, as described above.

That history is worth understanding, because it works differently from the SOL bot's. **Four parameters change without a restart:** TAKE_PROFIT, STOP_LOSS, GRID_SPREAD and GRID_RESET_THRESHOLD each have a **change** button on their card, and the engine picks the edit up within a few seconds. Everything else in `config_stocks.env`, capital included, still takes effect only when you `pm2 restart magellan-stocks`. Either way the timeline gains a row at the moment the engine adopts the change, badged so you can tell a hot reload from a start.

A hot reload **never moves a position the engine already holds**. Each unit's sell target and stop are stamped onto it when it buys, and those stored prices are what decide, so a new take-profit governs the next buy and leaves everything open exactly where it was. Changing the spread re-lays the resting buy ladder, which owns no tokens; the reset threshold applies from the next tick. The engine re-checks every value against the configuration it is actually running and refuses the whole edit if any of them no longer fits, so a change that would leave the deepest units stopping out unfilled is declined rather than half-applied.

There is no delete button on the history, unlike the SOL/USDC one: that file belongs to the stocks engine, and the dashboard only ever reads it.

One deliberate rule about the combined figures: **Today and Since Inception only add the stocks engine in when both engines are in the same mode**. If your SOL bot is live and the stocks engine is still in paper, the headline figures stay SOL-only and a note says so. Simulated money must never be summed into a number that reads as real. The trades themselves still appear in the feed and the History tab, each carrying its own PAPER or LIVE badge, because a record is not a claim.

Within that rule, everything is combined: Daily P&L, Transactions, Win Rate, Lifetime P&L, Completed Cycles, Days Active and **Transferred**, which counts what has left both beneficiary wallets rather than only the SOL one.

Annualized APY measures both engines against capital actually employed, not against today's figure. If you raise the stocks engine's `PRINCIPAL` from 500 to 1000, the profit it earned on 500 is not suddenly divided by 1000: it keeps a running total of principal multiplied by days, exactly as the SOL bot has always done, and the change is recorded at the restart that adopts it. The practical consequence is that adding capital to either engine no longer makes your past performance look worse than it was.

Unit Status shows both engines

On the **Trading** tab, **Unit Status** has one grid per engine. The SOL/USDC ladder is first and unchanged. Under it comes one labelled grid for each tokenized stock the engine holds, with the ticker, the company name, the grid centre, the last price it was evaluated against, and any status flag: **outranked** (no longer top-ranked, but the engine holds it and is still trading its ladder), **delisted**

(winding down: it keeps what it holds and buys nothing more), or **will not reopen** (closed against a premium and barred permanently). Hover any card for the entry, the take-profit, the stop-loss and the live unrealized P&L, exactly as on the SOL cards.

A stock unit card has a **sell now** button too, and it works the same way, with one difference that matters: a manual sell there IGNORES the mark guard. The engine defers a sell into a distorted peg on its own; when you click, it goes through anyway. The dialog says so before you confirm. And if the engine has not priced a name recently, the card says so and shows no P&L rather than computing one from a price of unknown age.

What can go wrong, and what it looks like

What you see	What it means
Free USDC reads <code>--</code>	the engine is in paper mode, so there is no on-chain balance
<code>EXIT_WHILE_PREMIUM</code> in the feed	a position closed against a rich peg; that name will not reopen
<code>SELL_DEFERRED</code> repeating	an exit has triggered but the peg is too distorted to sell into. A take-profit waits for the peg; a stop-loss is sent anyway after 30 minutes unless the engine is disarmed, and the alert says which of the two it is. It is written to the log every tick on purpose; the Telegram alert comes at most once every six hours per name
a stock unit card says <code>price unavailable</code>	the engine has not quoted that name in the last ten minutes, so it will not state a P&L it cannot stand behind
the whole panel is missing	<code>config_stocks.env</code> has no valid <code>OPERATOR_WALLET</code>
figures frozen with a stale badge	the <code>magellan-stocks</code> process is not running

Sizing your expectations

This engine trades a thinner market than SOL/USDC. Liquidity is lower, spreads are wider, and the tradable universe changes as names come in and out of the ranking, though a name you already hold does not leave when it drops out of the top: it keeps trading until it is flat. Start with a fraction of the capital you run on the SOL grid, and treat the first weeks as measurement rather than income.

[to top](#)

14. Quick Reference Cheat Sheet

Parameter relationships

```

TAKE_PROFIT > round-trip cost          → profitability requirement (enforced)
  (live: 2 × SLIPPAGE_BPS / 100 + 0.6, so 1.60% at 50 bps)
  (paper: 1.00% at SLIPPAGE_BPS=50 and PAPER_DEX_FEE=0.0025)
STOP_LOSS ≤ 3-4 × TAKE_PROFIT          → manageable loss ratio
GRID_SPREAD × active_units ≤ STOP_LOSS → grid fits inside one stop-loss (enforced)
GRID_RESET_THRESHOLD > GRID_SPREAD     → grid can reach its first target
(enforced)
GRID_RESET_THRESHOLD ≈ grid max depth   → prevents stale grids
  (grid depth = GRID_SPREAD × active_units)
MAX_DAILY_LOSS = your pain threshold    → capital protection

```

The three marked enforced are refused by the Settings dialog and at reload (see "The dashboard checks this for you"). `GRID_SPREAD=0` is exempt from both grid rules, and active units are `SPLIT_NUMBER` minus the buffer units (9 at `SPLIT_NUMBER=10`, `PRINCIPAL_LEFTOVER=10`).

Settings by market condition

Condition	TAKE_PROFIT	STOP_LOSS	GRID_SPREAD	GRID_RESET
Choppy / sideways	1.8%	2%	0.2%	2%
Volatile swings	2.0%	3%	0.3%	3%
Trending up	2.0%	2%	0.2%	1.5%
Trending down	2.0%	1.5%	0.15%	3%
Low volatility	2.0%+	3%	0.3%	3%

Minimum viable settings for live trading

```

MODE=live
TAKE_PROFIT=2.0      # clears the 1.60% live floor at SLIPPAGE_BPS=50
STOP_LOSS=2          # room to breathe, but not too much
GRID_SPREAD=0.2      # balanced density
GRID_RESET_THRESHOLD=2 # adapts to market movement
MAX_DAILY_LOSS=3     # conservative protection
PRINCIPAL_LEFTOVER=10 # safety buffer
HAPPY_GAIN=5         # regular profit extraction
SLIPPAGE_BPS=50      # standard Jupiter tolerance

```

The golden rules

1. **TAKE_PROFIT must beat your costs.** In live mode at the default `SLIPPAGE_BPS=50` the bot refuses 1.60% or less, so start at 2.0%.
 2. **Let MAX_DAILY_LOSS do its job.** Don't increase it after a bad day.
 3. **Paper test every parameter change** before applying it to live capital.
 4. **Check the dashboard daily.** Win rate, P&L trend, and stop-loss frequency are your vital signs.
 5. **Magellan loves oscillation.** Sideways choppy markets are where it shines. Trending markets require patience and protection.
 6. **Start small, scale gradually.** The paper → small live → full live pipeline exists for a reason.
 7. **Export and analyze your trades.** Data beats intuition every time.
-

Appendix A: Reading the SOL/USDC Operator Wallet panel

In live mode the Dashboard tab shows your operator wallet, split into what is tied up in open positions, what is committed to buys still to come, and what is spare. The units themselves are on the Trading tab.

Both wallet panels start collapsed, showing only their heading, the address and a **Show** control. They are the bulkiest things on the tab and the least often needed at a glance, so the page opens on the figures you read every time. Expand either one and Magellan remembers your choice in that browser; it never overrides you afterwards.

Five cards:

- **SOL in Positions:** the SOL your HOLDING units own, with its current market value underneath.
- **USDC in Positions:** what those positions *cost* you. Compare it against the value under the first card and the difference is your unrealized profit or loss on open positions.
- **USDC in Waiting:** USDC still in the wallet but already spoken for by units that have not bought yet. This is usually the largest of the five and it is not spare cash: it is what the grid will spend filling its remaining buy targets.
- **Free SOL:** SOL sitting in the wallet that no unit owns, with its market value underneath.
- **Free USDC:** what you could withdraw without starving the grid.

Free SOL is the one to keep an eye on. The bot will not sell it, because no unit is tracking it: it has no take-profit and no stop-loss, and it simply rides the price. That is not necessarily wrong (you may have deposited SOL deliberately), but it should never be a surprise. Before this panel existed, the only way to notice was to read a reconciliation line in the startup log.

Free USDC is not your wallet balance. It subtracts the USDC still set aside for units that have not bought yet, because that money is already spoken for. If you hold 314.95 USDC and three units are each waiting to spend 100, the USDC in Waiting card reads 300.00 and Free USDC reads 14.95, which is what you could actually take out and still let the grid do its job. Expect Free USDC to sit near zero when everything is funded; that is healthy, not a warning.

The USDC cards account for the whole balance. Free USDC is simply what is left of your wallet's USDC once USDC in Waiting is set aside, so the row tells you where all of it is rather than leaving you to take Free USDC on trust. The two cards will not always add up to the last cent on screen, because each is rounded down for display. USDC in Waiting counts units waiting to buy, units still idle (including any safety-buffer unit that never gets a target) and any buy that is mid-swap.

It does *not* subtract the USDC in your open positions, and that is deliberate: when a unit buys, its USDC is swapped away for SOL and leaves the wallet there and then. Subtracting it a second time would understate what you have by the whole deployed amount.

A figure shown in red is negative and worth investigating. Negative Free SOL means your units together claim more SOL than the wallet actually holds. Negative Free USDC means the grid does not have enough capital left to fill its remaining buy targets. Neither is rounded away to zero. Every card's tooltip spells out the arithmetic, so the numbers always reconcile against `checkBalances`.

The header shows how old the figures are. Balances refresh roughly every two and a half minutes, so an age of a few seconds to a couple of minutes is normal. If it climbs past five minutes it turns amber, which means the bot's balance fetches are failing: the panel deliberately keeps showing the last good numbers with a growing age rather than silently replacing them with something wrong.

The whole section disappears rather than show a figure it cannot stand behind: in paper mode (there is no wallet), before the bot's first balance fetch after a restart, or whenever a balance or price reads as corrupt.

Appendix B: Withdrawing money to your beneficiary wallet

Under **Free SOL** and **Free USDC** there is a **Withdraw** button. It is the answer to the obvious question the panel raises: fine, that money is spare, how do I get it out?

Clicking it opens a dialog that shows the whole sum: what is in the wallet, what your open positions own, what is being kept back for gas, and therefore what is available. Type an amount, or press **MAX** to take everything available. **Review** then shows a confirmation screen with the destination address and what will be left behind. Nothing moves until you click **Confirm withdrawal**.

Funds always go to the `BENEFICIARY_WALLET` in your `config.env`. You cannot type a different address into the dashboard, and that is on purpose: a withdrawal screen that accepts an arbitrary destination is the single most useful thing an attacker could find on a compromised dashboard.

Some SOL always stays behind. The bot needs SOL to pay for its own transactions, so a SOL withdrawal keeps back 0.1 SOL, or a little more than your `SOL_GAS_AMOUNT` if you have set that above 0.1. The "a little more" matters: the bot's own gas check wants slightly more than the threshold, so leaving exactly `SOL_GAS_AMOUNT` would have it halt itself the moment the withdrawal landed. MAX therefore never empties the wallet and never stops your bot. USDC has no such reserve: the Free USDC figure is already what is spare.

The amount you confirm is a ceiling, not an instruction. The dashboard's figures are up to a couple of minutes old, so before sending anything the bot re-reads your wallet on chain. If you pressed MAX and the free balance has fallen since, it sends the smaller amount and tells you it did. If you typed an exact number that no longer fits, it sends nothing rather than quietly sending less. It can never send more than the number you saw.

Available is not quite the same as Free. The withdrawal figure is slightly more cautious than the Free SOL card above it, in two cases: while a unit is mid-sell, its SOL still counts as committed even though the panel already shows it as free, and if the bot recovered a position after a restart without being able to work out how much SOL it holds, SOL withdrawals are blocked entirely until that resolves. Being cautious costs you a few minutes; being wrong costs you the SOL.

One at a time. From the moment you confirm until the transfer reaches a definite outcome, both buttons stay disabled and a second request is refused, and that hold covers the whole transfer however long the queue wait before it was. That covers the transfer itself, not just the wait before it: a transfer can take up to about a minute, during which the dashboard still shows the old balance, and the natural reaction to "nothing happened" is to try again. That is exactly how people send money twice.

Withdrawing USDC changes two other numbers. **Transferred** in Since Inception goes up, because it now counts both the automatic HAPPY_GAIN transfers and anything you take out by hand. And the bot's running profit total comes down, because a withdrawal spends profit first, exactly the way HAPPY_GAIN does. So if you take out less than the bot has earned, you are moving money your Lifetime P&L has already counted. If you take out more, the rest comes out of your principal: the confirmation screen tells you how much, and you should lower **PRINCIPAL** in Settings by that amount afterwards. If you do not, the bot's startup balance check will report a mismatch every time it restarts, because it is still expecting capital you have taken home.

Your Lifetime P&L itself does not move when you withdraw. Taking money out is not a trading result, and it would be strange if moving your own money counted as a loss. What changes is how much of that profit is still sitting in the wallet rather than in your beneficiary account.

Every withdrawal, successful or not, appears in **Trade History** with the asset in the Unit column and a Solscan link. A successful one also sends you a Telegram message. A failed one does not, on the assumption that you are looking at the dashboard at the time.

You can withdraw while the bot is halted, which is usually the order you want: stop trading, then take the money out.

Appendix C: Selling a position by hand

Magellan exits a position on exactly two triggers, and both are decided the moment it buys: the take-profit above your entry, and the stop-loss below it. Most of the time that is exactly what you want, because the whole point of a grid bot is that it does not panic and does not get bored. But sometimes you know something the bot does not. A listing gets announced, a chain halts, the market turns while one unit sits 3% underwater with a take-profit it will not reach this week. Until now the only answers were to lower `TAKE_PROFIT` for every position at once, or halt the bot, which stops it buying but does not close anything.

Every HOLDING unit card now carries a red sell now button. Click it and a dialog shows you the position: how much SOL it holds, what you paid, what the price is now, your profit or loss in both percent and USDC, where its take-profit and stop-loss sit, and what it cost you. These are the same numbers you get by hovering the card, gathered in one place so you are deciding against the full picture rather than a badge and a price. If the position is down, the dialog says so plainly, because that is the case where selling costs you something real.

Review shows a confirmation screen and nothing moves until you click Confirm sell. That screen repeats the profit or loss you are about to realize and warns you that the sale cannot be undone. Escape or a click outside takes you back a step rather than closing everything, so a stray keypress never cancels work you meant to do.

A few things are worth understanding before you use it.

The unit goes back to work afterwards. A manual sell is an ordinary sell in every respect: the unit returns to IDLE, picks up a new buy target on the next grid assignment, and may well buy again within minutes if the price comes to it. Selling by hand closes a position, it does not retire a unit. If you want the capital to stop trading entirely, halt the bot or wind it down.

The price you see is the last one the bot recorded, not a quote. The sale fills at whatever the market gives when the bot runs it, usually within one tick. On a fast-moving chart the fill can differ from the figure on screen. This is the same deal every other trade gets, and it is why the dialog says "at about" rather than promising a number.

A manual sell at a loss is not a stop-loss. This matters more than it sounds. The circuit breaker watches for a run of stop-losses in a short window and pauses buying when it sees one, because a cascade of stop-outs usually means the market has moved against the whole grid. If closing three positions by hand counted as three stop-losses, you would trip that breaker yourself and stop the bot buying on units you never touched. So manual sells are recorded separately, as `MANUAL SELL EXECUTED` in Trade History, and they never feed the circuit breaker or the post-stop-loss cooldown. They do count in your P&L, your trade count and your win rate, because they are real trades.

The position is checked again before anything is sold. The dashboard cannot sell anything by itself: it queues a request and the bot acts on its next tick. In between, that unit might hit its take-profit and buy back in, which would leave you selling a position you never looked at. So the bot re-checks that the position is still the same one the dialog described, down to the exact entry price, and refuses if it is not. You will see `MANUAL SELL FAILED` with the reason instead of a sale.

One at a time, and the bot has to be running. A second request is refused rather than replacing the first, so you cannot queue two sells and be unsure which one closes. If the bot is stopped, the dashboard refuses to queue at all rather than leaving a request that would quietly expire.

You can sell by hand while the bot is halted. That is often the order you want: stop the bot buying, then close the positions you no longer like at your own pace.

Every manual sell sends a Telegram message, one per unit, so closing several in a row gives you several confirmations rather than one. It works in paper mode too, which is the sensible place to try it before you use it on real money.

Appendix D: Wind-Down Mode

Wind-down mode lets you gracefully exit all positions and withdraw your capital. Unlike the Emergency Stop, which halts everything immediately, wind-down is a patient exit: existing positions sell at their normal take-profit or stop-loss targets, but no new buys are placed.

What happens during wind-down

1. **New buys stop immediately.** All `WAITING_TO_BUY` units are cancelled and moved to `IDLE`. No new buy targets are assigned.
2. **Existing positions sell normally.** Units currently `HOLDING SOL` continue to monitor for take-profit and stop-loss exits. They sell at their original targets, not at market price.
3. **Grid resets are disabled.** No point recalculating buy targets that will never be filled.
4. **Capital returns to USDC.** As each unit sells, it returns to `IDLE`. Once all units are idle, your full `PRINCIPAL` is back in `USDC` and safe to withdraw.

How to activate

Three ways to trigger wind-down, all doing the same thing:

Method	How	When to use
Dashboard	Toggle the trading switch to "Winding Down"	You are at your computer
Telegram	Send <code>/halt</code> to the bot	You are away from the dashboard
SSH	<code>touch HALT</code> in the bot directory	Fallback if dashboard or Telegram are down

All three methods create or remove the `HALT` file. The underlying mechanism is the same regardless of how you trigger it.

It halts both engines

If you run the tokenized-stocks engine, the same switch stops that too. One `HALT` file, both engines, one meaning: no new buys anywhere, and open positions still sell at their targets on both sides. Removing it resumes both.

You will get **two** Telegram alerts, one from each engine, each saying which it is. That is deliberate: it is the only way to know the second engine actually saw the halt, and it still works if one of them happens to be stopped.

The stocks engine has no other live off switch. `TRADE_STOCK` in either config file is read once at startup, so changing it needs `pm2 restart magellan-stocks`; `HALT` is the one that acts on a running process.

Monitoring progress

While winding down, the dashboard shows two indicators:

- **Red banner:** "Trading halted by operator" (confirms wind-down is active)
- **Yellow banner:** "X of Y units still holding SOL. Waiting for exits..." (live progress counter)

If the stocks engine is running, the yellow banner counts both, reading something like "Still holding: 2 of 9 SOL units and 3 of 6 stocks units". It turns green only when **both** are empty.

Two cases it deliberately refuses to call complete, because "Safe to withdraw" over an open position is the worst thing it could say: a stocks engine that has stopped answering (its last positions are unknown, not closed), and one that is running but has not registered the halt yet (it can still buy). Both say which, in the banner.

When all positions have closed:

- The yellow banner turns **green:** "All positions closed. Safe to withdraw."
- A **Telegram alert** is sent: "All positions closed. No SOL held. Safe to withdraw."

The completion alert only fires after 3 consecutive ticks with no active positions, preventing false positives from transient state transitions.

How to resume trading

To cancel wind-down and resume normal trading:

- **Dashboard:** toggle the switch back to "Trading"
- **Telegram:** send `/resume`
- **SSH:** `rm HALT`

The bot immediately re-assigns buy targets at the current market price and resumes normal grid trading.

How long does wind-down take?

It depends on how many units are HOLDING and how far they are from their take-profit targets. In a choppy market, most positions resolve within hours. In a strong downtrend, positions may hit their stop-loss instead, which closes them faster but at a loss.

You can check progress at any time via the dashboard banner, or send `/status` on Telegram.

Telegram commands

Command	Action
<code>/halt</code>	Stop trading (enter wind-down mode)
<code>/resume</code>	Resume trading (exit wind-down mode)
<code>/status</code>	Bot status, positions, price, P&L, uptime

Wind-down vs stopping the process

	Wind-Down (HALT file)	Process Stop (pm2 stop)
New buys	Stopped	Stopped (process not running)
Open positions	Sell at normal take-profit/stop-loss targets	Nothing happens (process not running)
State preserved	Yes, <code>state.json</code> stays current	Yes, last saved state remains on disk
Trigger	Dashboard / Telegram / <code>touch HALT</code>	<code>pm2 stop magellan</code> or Ctrl+C
Use case	Graceful exit, planned withdrawal	True emergency, server maintenance

When to use which: Wind-down is the normal way to exit. It keeps the bot running so positions close at their targets. Only stop the process if you need everything to halt immediately (e.g., suspected bug, server migration). If you stop the process while units are HOLDING, the bot picks up where it left off on restart.

Appendix E: Telegram Bot

Magellan sends you a Telegram message every time something meaningful happens. You can also send commands back. No need to open the dashboard.

Alerts you will receive

Alert	Emoji	When it fires
Bot started	🟢	Bot starts up
Bot stopped	🔴	Bot shuts down gracefully
Stop-loss triggered	📉	A unit sold at its stop-loss level
Buy abandoned	❌	Buy failed after all retries exhausted
Sell abandoned	❌	Sell failed after all retries exhausted
Profit transfer	💰	HAPPY_GAIN threshold reached, USDC moved to reserve wallet
Daily loss limit	🛑	Daily loss threshold exceeded, trading halted
Low gas	💸	SOL balance too low for transaction fees
Operator halt	🛑	Kill switch activated
Operator resumed	▶️	Kill switch deactivated
Wind-down complete	✅	All positions closed, safe to stop the bot
Config reloaded	⚙️	A hot-reload parameter was changed without restarting
Positions retargeted	🎯	Open positions were moved to a new take-profit
Circuit breaker activated	⚡	Stop-loss cascade detected, buys paused until cooldown expires
Circuit breaker expired	▶️	Cooldown expired, normal trading resumed
Circuit breaker sell-only	🔴	Max triggers reached, sell-only for rest of day
Daily summary	📊	End-of-day summary with trade counts and P&L

Alerts from the tokenized-stocks engine

That engine sends its own messages, each prefixed **Stocks** so a shared chat makes it obvious which engine spoke. It has no commands of its own: everything below is an alert.

Alert	Emoji	When it fires
Engine started / stopped	▶ / ◻	The stocks process starts or shuts down
Trading disarmed	🔒	It is running with <code>TRADE_STOCK</code> off: screening only, no orders
Operator halt / resumed	🛑 / ▶	The shared kill switch, so you get one alert from each engine
Wind-down complete	✅	Every stock position closed while halted
Bought / sold	🟢 / 🔵	A fill, with the amount, the price and the discount to the mark
Stop-loss	🔴	A stock unit sold at its stop-loss
Sold at a premium	🔵	A winning exit taken while the token was rich versus its mark. Expect this on most profitable stock trades: a take-profit fires when the token rises, and a token outrunning its mark IS a premium. The name is not reopened until the peg is back inside the band
Sell held back	⏸	A take-profit fired but the peg was too distorted to sell into. Sent at most once every six hours per name, because it is a standing condition rather than an event
Buy / sell failed	❌	All retries exhausted
Daily loss limit, circuit breaker, low gas	🔴 / 🚨 / ⚡	Each sent once when the condition starts, and once when the circuit-breaker cooldown expires
Withdrew / withdrawal refused	👉 / ❌	A manual withdrawal from the stocks wallet

Commands you can send

There are 13, and **every one of them now answers for both engines**. No command was added for the stocks engine: Magellan runs one Telegram connection, and a second one would split the message stream between two processes.

Command	What happens	What the stocks engine adds
<code>/status</code>	Bot status, positions, price, P&L, uptime	Its mode, how many grids it runs, its unit states and today's P&L. While halted it also says whether it has registered the halt and how far its wind-down has got
<code>/pnl</code>	Daily and lifetime P&L report	Its own block, then a combined total, but only when both engines are in the same mode
<code>/balance</code>	Operator wallet balances	The stocks wallet, as that engine last recorded it, with the age stated
<code>/config</code>	Current hot-reload parameters with units	Its capital and trade parameters, marked with which hot-reload
<code>/cb</code>	Circuit breaker status and cooldown	Its own breaker and cooldown
<code>/units</code>	Per-unit breakdown: state, prices, unrealized P&L, SL cooldown remaining	Its units grouped per stock, with the grid centre, the last price and any outranked, delisted or will-not-reopen flag
<code>/halt</code>	Stop trading on both engines (enter wind-down mode)	Reads the same file
<code>/resume</code>	Resume trading on both engines (exit wind-down mode)	Reads the same file
<code>/vps</code>	VPS health: disk, RAM, CPU, logs, PM2, NTP	Whether the stocks engine is still writing its state file
<code>/sol_price</code>	Current SOL price with its 24h and 7d change, plus whether price updates are on and when the next one is due	Nothing: these three are about the market, not about either bot
<code>/sol_start</code>	Start sending the SOL price and its 24h and 7d change every 30 minutes (replies straight away)	
<code>/sol_stop</code>	Stop the 30-minute SOL price updates	
<code>/help</code>	List all available commands	

Watching the SOL price from Telegram

The alerts above all describe the bot. If you also want to keep half an eye on the market itself, send `/sol_start`: Magellan will message you the current SOL price every 30 minutes, with how far it has moved over the last 24 hours and the last 7 days until you send `/sol_stop`. The reply to `/sol_start`

carries the price straight away, so you can see immediately that it is working, and `/sol_price` gives you the price on demand at any time along with whether updates are running and how long until the next one.

The price itself costs nothing extra: the bot already fetches and caches it on every loop to run the grid, so the updates reuse a number it already has. The 24h and 7d changes are the one thing that is not already on hand, so Magellan looks them up separately, at most once an hour and only while you are actually using the feature. Leave the updates off and never send `/sol_price` and it makes no extra call at all.

A few things worth knowing:

- **Underscores are the real command names.** Telegram only registers commands made of letters, digits and underscores, so `/sol_start` is what appears in the bot's autocomplete menu and what you can tap. `/sol-start`, `/sol-price` and `/sol-stop` are accepted too, since the hyphen is the natural way to type them.
- **The setting survives a restart.** If you deploy an update or the VPS reboots, the updates you switched on stay switched on. A bot that was off for hours sends one catch-up message when it comes back, not one for every interval it missed.
- **They keep coming while the bot is halted.** Wind-down, a daily loss halt and a circuit-breaker cooldown do not stop them, which is usually when you most want to watch the price by hand.
- **They pause during a price-feed outage.** If the price API is failing, Magellan goes quiet rather than repeating its last known price as though it were live. `/sol_price` will say the feed is degraded and tell you how old the number actually is.
- **The 24h and 7d changes go quiet rather than guessing.** They come from a separate price source, and if it is unreachable or its last answer has gone stale, Magellan simply leaves them off the line and sends the price on its own. You will never see a placeholder or a figure it is not sure of.
- **The change is measured against the price in the same message.** Both figures are recalculated at the moment the message is written, so the price and the percentages beside it always describe the same instant.
- **This is not a price alarm.** There is no target price to set and nothing fires when SOL crosses a level: it is a plain readout on a timer.

A few things about the replies

- **Long replies arrive as several messages.** Telegram refuses anything over 4096 characters, so `/units` with both engines full is split at line boundaries and sent in order.
- **"Today" says -- when it cannot be trusted.** Only a running engine rolls its daily counters over at midnight UTC. If a bot is stopped, its previous day's figures are still sitting in the state file, so Magellan withholds them rather than presenting yesterday under a heading that says Today.
- **`/balance` reads the stocks wallet from that engine's own records, not from the chain.** It refreshes them itself every few minutes, and the reply says how old they are. Only the SOL/USDC balances are read live, because that is the wallet the bot you are talking to owns.
- **A combined P&L appears only when both engines are in the same mode.** A live bot beside a paper stocks engine is a perfectly normal setup, and adding simulated money to a real figure would make it read as real. You still see both blocks; you just do not see them added together.

Note on timing: Alerts are sent instantly when the event happens. Commands are processed within a few seconds as the bot polls Telegram on a short interval. There may be a brief delay before you see the effect in the dashboard.

Appendix F: How to Change Capital and Other Structural Parameters

Structural parameters define the shape of Magellan's trading grid: how many units exist and how much each one spends.

***This appendix is about the SOL/USDC bot.** The tokenized-stocks engine is a separate process with its own `config_stocks.env`, no hot reload of capital and no dashboard resize, and it has a third structural parameter (`STOCK_COUNT`) that the SOL bot does not have. Its procedure is at the end of this appendix, under "Changing capital on the tokenized-stocks engine". Following the dashboard steps below for the stocks engine will not work: there is no control for it.*

For the SOL/USDC bot, `PRINCIPAL` and `SPLIT_NUMBER` can be changed from the dashboard, without stopping the bot, rebuilding anything, or deleting `state.json`. `PRINCIPAL` alone can be changed while the bot is trading; `SPLIT_NUMBER` needs the bot halted with every unit idle. The rest still need a restart.

Which parameters are structural?

Parameter	What It Controls	Default	How to change it
<code>PRINCIPAL</code>	Total USDC budget allocated to trading	required (no default)	Dashboard, Settings tab
<code>SPLIT_NUMBER</code>	Number of trading units (grid slots)	required (no default)	Dashboard, Settings tab
<code>PRINCIPAL_LEFTOVER</code>	% of principal reserved as safety buffer	<code>10</code>	Restart
<code>MODE</code>	<code>paper</code> (simulated) or <code>live</code> (real swaps)	<code>paper</code>	Restart
<code>MIN_TRADE_USDC</code>	Minimum trade size in USDC	<code>1</code>	Restart
<code>RPC_URL</code>	Solana RPC endpoint	-	Restart
<code>FALLBACK_RPC_URL</code>	Backup RPC endpoint	-	Restart
<code>JUPITER_API_URL</code>	Jupiter swap API base URL	<code>https://api.jup.ag/swap/v2</code>	Restart
<code>JUPITER_API_KEY</code>	Jupiter API key (required for live mode)	-	Restart
<code>OPERATOR_WALLET_PRIVATE_KEY</code>	Base58 wallet private key	-	Restart
<code>BENEFICIARY_WALLET</code>	Public key for profit transfers	-	Restart

Everything else is one of the 14 hot-reloadable tuning parameters (`TAKE_PROFIT` , `STOP_LOSS` , `GLOBAL_SL_COOLDOWN_MIN` , `GRID_SPREAD` , `GRID_RESET_THRESHOLD` , `MAX_DAILY_LOSS` , `SOL_GAS_AMOUNT` , `SLIPPAGE_BPS` , `HAPPY_GAIN` , `POLL_INTERVAL_MS` , `CIRCUIT_BREAKER_COUNT` , `CIRCUIT_BREAKER_WINDOW_MIN` , `CIRCUIT_BREAKER_COOLDOWN_MIN` , `CIRCUIT_BREAKER_MAX_TRIGGERS`), which you can change at any time while the bot is trading.

Your statistics are never affected

This is worth stating plainly, because older versions of this guide told you to delete `state.json` , and that **did** destroy them.

Changing capital replaces one thing: the list of grid units. It does not touch your inception date, lifetime P&L, completed cycles, win rate, transferred total, or any daily counter. Your JSONL trade logs under `log/` are append-only and are never rewritten, so the History tab is unaffected in any case.

Your statistics do not move, including the percentages. Lifetime P&L, completed cycles, win rate, days active and transferred total are absolute counters: they never involve `PRINCIPAL`, so a capital change cannot touch them. Annualized APY is a percentage and therefore could, but Magellan measures it against your **average capital employed** rather than today's `PRINCIPAL`. Trade 500 USDC for 100 days and then 1000 for 100 days, and the denominator is 750, not 1000. The return you earned on 500 stays attributed to 500.

Earlier versions divided lifetime profit by whatever `PRINCIPAL` happened to be today, so doubling your capital halved the reported APY overnight even though nothing about the trading had changed.

One deliberate exception: the Dashboard tab's **Daily Return** stays measured against current `PRINCIPAL`, because both halves of it are today's and a capital change cannot distort it. The **History tab's Daily Return for a past date** divides by the `PRINCIPAL` that was running on that day, read from the Configuration History, so a capital change does not re-base older days.

When capital can be changed

`PRINCIPAL` on its own: any time, while the bot trades. Changing it only changes how much each future buy spends. Units that have not bought yet (waiting for their price, or idle) take the new size on the bot's next tick. Units holding SOL keep the cost they actually paid, and their take-profit and stop-loss do not move; they take the new size once they sell and come round again. Nothing is halted and nothing is sold.

Raising `PRINCIPAL` is always allowed. Lowering it is allowed down to the USDC already in open positions, and no further: a `PRINCIPAL` smaller than what your positions cost describes a grid smaller than the one your wallet is carrying. If three positions cost 300 USDC between them, 300 is the lowest `PRINCIPAL` you can set until some of them close.

`SPLIT_NUMBER` : only on a quiet grid. Changing the unit count rebuilds the unit array, and a unit holding SOL cannot simply be deleted or resized: it owns a real position with a real entry price, take-profit and stop-loss. So Magellan requires the grid to be **wound down** first: the bot halted, and every unit idle.

That is not an extra chore, because it is the same wind-down you already do before withdrawing. Halt the bot and it cancels every waiting buy target and lets open positions sell out on their own terms. When the banner turns green and reads *"All positions closed. Safe to withdraw."*, the grid is quiet.

Both rules are enforced three times over: the confirmation screen disables **Apply** and tells you why, the server refuses the change, and the bot re-checks before changing anything. The **change** button itself is only locked while the bot is not responding.

Changing PRINCIPAL from the dashboard

Step 1: Open Settings and find the Capital group. The **change** button sits on the Grid Capital frame.

Step 2: Enter the new PRINCIPAL and leave SPLIT_NUMBER as it is.

Step 3: Review the confirmation screen. It shows what each buy will spend before and after, how many open positions keep their entry cost and how much USDC they committed, and "Trading continues." (or "Trading stays halted; buys use the new size once it resumes." when the bot is halted). It also warns you if the wallet holds less USDC than the new PRINCIPAL, and if a cut would trip today's daily loss stop at once (see below).

Step 4: Apply. Within a few seconds the waiting units carry the new size. Nothing else changes and nothing pauses.

Lowering PRINCIPAL moves the daily loss stop at once. MAX_DAILY_LOSS is a percentage of PRINCIPAL, so a smaller PRINCIPAL means a smaller stop, applied to today's realized P&L the moment the change lands. If today's losses already exceed the new stop, buying stops for the rest of the UTC day (open positions can still sell). The confirmation screen tells you when that would happen.

Changing SPLIT_NUMBER from the dashboard

Step 1: Halt the bot. Use the kill switch in the header. The chip changes to PAUSED and the bot stops opening new positions.

Step 2: Wait for the wind-down. Open positions sell at their own take-profit or stop-loss, so this takes anywhere from moments to hours depending on the market. The banner counts down for you (2 of 5 units still holding SOL). Nothing is at risk while you wait; the bot is simply not buying.

Step 3: Open Settings and find the Capital group. Press **change** on the Grid Capital frame.

Step 4: Enter the new values. You will see:

- how many units you will have, split into active and buffer
- **what each buy will actually spend**, which is the number that matters
- `PRINCIPAL_LEFTOVER`, shown read-only, because it is what explains the unit size
- a warning if `PRINCIPAL` exceeds the USDC actually in your wallet
- how the daily loss stop moves, since `MAX_DAILY_LOSS` is a percentage of `PRINCIPAL`

If the grid is not quiet yet, the confirmation screen says what is still outstanding (the bot is not halted, some units have not closed yet, or the circuit breaker is active) and **Apply** stays disabled.

Step 5: Review and Apply. The confirmation screen freezes the values you reviewed, so nothing typed afterwards can change what gets sent.

Step 6: The grid rebuilds within a few seconds. Check the unit count and per-unit size on the Trading tab.

Step 7: Remove the halt when you are ready. Nothing trades until you do. Buy targets are assigned from the current price the moment you resume, so you control exactly when the new grid goes to work.

Every capital change is recorded

The Settings tab's **Configuration History** timeline gets a new entry the moment the change lands, tagged `capital change` so you can tell it apart from an ordinary restart or a hot reload. Expand the row and it shows `PRINCIPAL 500.00 USDC -> 1,000.00 USDC` and `SPLIT_NUMBER 5 -> 10` highlighted against the previous snapshot, with every other parameter listed unchanged beside them.

You also get a Telegram alert saying what changed. For a `SPLIT_NUMBER` change it names the new grid (how many units, how many are active, what each one will spend) plus a reminder that trading stays paused until you remove the halt. For a `PRINCIPAL`-only change it says what each buy now spends, how many units took the new size, how many open positions keep their entry cost, and that trading continues, or, if the bot is halted, that trading stays halted and buys use the new size once it resumes. The same summary is written to `error.log`, so there is a record even if you were not watching.

Nothing about this is retroactive. Snapshots taken before a change keep the values that were in force at the time, which is what makes the timeline a usable audit trail of when your capital actually moved.

***If you are raising `PRINCIPAL` in live mode, deposit the extra USDC first.** The dashboard warns you when the new `PRINCIPAL` exceeds your wallet balance, but it does not stop you. Trading with less capital than `PRINCIPAL` claims leaves the deepest buy levels unfillable.*

Why "each buy" is not PRINCIPAL divided by units

This surprises most operators the first time, and the confirmation screen spells it out for exactly that reason.

`PRINCIPAL_LEFTOVER` holds back a safety buffer. When that buffer works out to a whole number of units, those units simply sit idle and every active unit gets the full share. When it does not, there are no buffer units at all and the buffer comes out of every unit instead.

With `PRINCIPAL=500`, `SPLIT_NUMBER=5` and `PRINCIPAL_LEFTOVER=10`, the buffer is half a unit, which rounds down to none. All 5 units are active, and each spends $(500 - 50) / 5 = 90$ USDC, not 100.

The Settings card shows the per-unit share of `PRINCIPAL`; the confirmation screen shows what a buy actually spends. When they differ, trust the second one.

Changing parameters that still need a restart

For `PRINCIPAL_LEFTOVER`, `MODE`, `MIN_TRADE_USDC`, the RPC and Jupiter settings, or the wallet keys:

```
cd /var/www/magellan/current
touch HALT # let open positions wind down first
cp state.json "state.json.pre-change-$(date -u +%Y%m%dT%H%M%S).bak"
nano config.env
npm run build && node dist/tools/validateConfig.js
pm2 restart magellan
rm -f HALT
```

Do not delete `state.json`. The bot reconciles the grid against your config on every restart: it adds or removes idle units, and it never touches a unit that is holding SOL. Deleting the file only throws away your statistics.

Editing config.env directly

The bot watches `config.env`, so an SSH edit works for the tuning parameters too. For `PRINCIPAL` and `SPLIT_NUMBER` the same rules apply as from the dashboard: a `PRINCIPAL`-only edit applies while trading unless it cuts below the USDC in open positions, and a `SPLIT_NUMBER` edit needs the wound-down grid. When the rule is not met, the bot logs that it declined the change and keeps running as it was. The value is not lost, it simply waits for the next restart. To apply it sooner, meet the rule (for `SPLIT_NUMBER`, halt and let the grid wind down; for a `PRINCIPAL` cut, let positions close or pick a larger figure), then make the file change again: saving the same numbers a second time reads as no change and applies nothing. Set the values back to what the bot is running and save, wait about ten seconds for the bot to read that save (it checks the file every 5 seconds, and this step logs nothing), then enter the new values and save once more. A restart also applies it.

Common resize scenarios

Adding capital. Raise `PRINCIPAL` ; on its own this applies while trading. Keep `SPLIT_NUMBER` the same for bigger positions at the same grid depth, or raise both to keep position size steady across more units. More active units at the same `GRID_SPREAD` make the grid deeper, and the bot refuses a grid deeper than `STOP_LOSS` , so lower `GRID_SPREAD` beforehand if $\text{GRID_SPREAD} \times \text{new active units}$ would exceed it. Deposit the USDC first.

Reducing risk. Lower `PRINCIPAL` ; on its own this applies while trading, down to the USDC in open positions. Lower it first, then withdraw the surplus from the Dashboard tab: the waiting units shrink as soon as the change lands, which is what frees the USDC the withdrawal ceiling counts as yours. Keep `PRINCIPAL` matching what remains, or the startup balance audit reports a discrepancy on every restart.

Trading more, smaller units. Raise `SPLIT_NUMBER` alone. More units, each smaller, on denser price levels: the grid can never be deeper than `STOP_LOSS` ($\text{GRID_SPREAD} \times \text{active units}$), so lower `GRID_SPREAD` first until it fits the new unit count, or the change is refused (at `STOP_LOSS=2` , going from 10 to 12 units means 11 active, which needs `GRID_SPREAD` of 0.18 or less). Watch the per-unit size in the preview: it falls proportionally, and if it drops below `MIN_TRADE_USDC` the change is refused.

Taking bigger positions. Lower `SPLIT_NUMBER` alone. Fewer, larger units covering a narrower range.

Pre-change checklist

- For a `SPLIT_NUMBER` change: bot halted and the wind-down banner is green (a `PRINCIPAL` -only change needs neither)
- Lifetime figures noted, so you can confirm afterwards that they did not move
- For a `PRINCIPAL` increase in live mode, the USDC is already in the wallet
- The per-unit size in the preview is comfortably above `MIN_TRADE_USDC`
- You understand that your statistics, APY included, are not distorted by the change

Structural parameters FAQ

Q: Do I lose my statistics?

No. `state.json` is never deleted and the lifetime counters are never touched. Note the figures before you start and compare afterwards if you want to see it for yourself.

Q: Does my APY drop after I add capital?

No. Annualized APY is measured against your **average capital employed**, not against today's `PRINCIPAL`, so profit earned on a smaller grid stays attributed to that smaller grid. Trade 500 for 100 days then 1000 for 100 days and the denominator is 750. Lifetime P&L, completed cycles and win rate never involve `PRINCIPAL` at all and cannot move.

Q: Is any percentage still measured against today's PRINCIPAL?

One, deliberately. The Dashboard tab's **Daily Return** compares today's P&L with today's capital, so a change cannot distort it. The **History tab's Daily Return** for a past date uses the `PRINCIPAL` that was running that day, so older days keep their figures after a capital change.

Q: Can I change PRINCIPAL alone?

Yes, and without halting. The unit count stays the same. Every unit that has not bought yet takes the new per-unit amount at once; units holding SOL keep their entry cost and take the new amount when they next cycle.

Q: Do I have to wait for open positions to close?

Not for `PRINCIPAL` alone, unless you want to lower it below the USDC those positions cost. For `SPLIT_NUMBER`, yes, and the bot will not let you skip it: halt and let them sell at their own targets. Selling by hand at a loss to hurry a config change is rarely a good trade.

Q: What if I change my mind while waiting for a wind-down?

Remove the halt. The bot reassigns buy targets from the current price and carries on as before.

Q: Does it need a restart?

No. Not for `PRINCIPAL` or `SPLIT_NUMBER`. The running bot picks the change up on its next tick.

Q: Can I edit state.json manually?

No, and there is no reason to. The bot owns that file and rewrites it every tick, so a hand-edit races with it. The rebuild does the same job correctly, at the one moment nothing is at stake.

Q: What about settings_history.json?

Keep it. A new `CONFIG_SNAPSHOT` is appended with the new parameters, and the Settings tab shows the change in the configuration history timeline.

Changing capital on the tokenized-stocks engine

The stocks engine hot-reloads only `TAKE_PROFIT`, `STOP_LOSS`, `GRID_SPREAD` and `GRID_RESET_THRESHOLD`. Everything else in `config_stocks.env` is read **once, at startup**, and there is no dashboard control for capital, so a restart is how capital changes, and it is also the only moment the grids are reshaped. All three of `PRINCIPAL`, `SPLIT_NUMBER` and `STOCK_COUNT` are changed the same way.

You do not need to wind down first, and you must not delete `state_stocks.json`. Open positions are kept, their entry, take-profit and stop-loss are never rewritten, and every lifetime counter survives.

```
cd /var/www/magellan/current
node dist-stocks/tools/validateConfig.js      # 1. check the NEW numbers before you
rely on them
pm2 stop magellan-stocks                      # 2.
nano config_stocks.env                       # 3. edit PRINCIPAL / SPLIT_NUMBER /
STOCK_COUNT
node dist-stocks/tools/validateConfig.js      # 4. check them again, offline, before
restarting
pm2 start magellan-stocks                     # 5.
pm2 logs magellan-stocks --lines 30          # 6. confirm what it came back as
```

Step 4 is the one worth not skipping. It is offline and instant, it prints the new unit size and the round-trip cost, and it refuses the combinations the engine itself refuses, so you find out before the restart rather than after it.

What to expect in the log. `ENGINE_STARTED` states the shape it actually came up with, in the form `3 names x 5 units at 66.67 USDC (principal 1000)`. If the grids were reshaped you also get a `GRID_RESIZED` line saying how many units were added, removed or resized.

Raising `PRINCIPAL` raises the size of every unit that has not bought yet. Units already holding a position keep the cost they actually paid, which is what keeps their profit and loss honest, so the grid runs at mixed sizes until those positions close. Your APY is not re-based by the change: the engine records capital over time, so profit earned on the old capital is still measured against the old capital.

Raising `SPLIT_NUMBER` adds units. **Lowering it** removes idle ones from the end; units that are busy cannot be removed yet, so they stay, keep their old size, and are cleared at the next restart once they are idle. The log line tells you how many were blocked.

Raising `STOCK_COUNT` lets the engine open more names, which it does on the first tick after the restart. **Lowering it** does not close anything: a name the engine is holding keeps trading until it is flat, and only then is its grid collected. Until that happens you will have more grids than `STOCK_COUNT`, which is expected and is stated in the `GRID_RESIZED` line. While it lasts, the unit sizes are never raised to the new value, so nothing over-commits. A size that falls, because `PRINCIPAL` was cut as well, is applied.

Buying pauses if the change puts more money at risk than the new `PRINCIPAL` allows. Cutting capital while positions are open is the ordinary way to reach this. The engine keeps managing what it holds and resumes buying as positions close; the reason appears in the log as `over_deployable_capital`.

If you make a mistake in the file, the engine does not die. It starts on the last configuration it recorded, sends a Telegram alert saying so, and runs **disarmed**: your positions keep their take-profit and stop-loss, withdrawals and manual sells still work, and it opens nothing new until you fix the file and restart. The Configuration History in the Settings tab marks that start as a fallback rather than showing a row identical to the one before it.