Zion Core — Technology Architecture Overview

Zion Core is built as a Python-based automated futures trading platform deployed on Railway. The backend runs as a FastAPI application inside a Docker container and manages user-specific trading engines, market monitoring, execution, notifications, analytics, and billing.

Core Backend

The system runs as a centralized backend service rather than a separate deployment for every user. Each user can have an individual trading engine with their own API keys, trading settings, risk limits, Telegram configuration, and active trades.

This allows Zion Core to support multiple users while keeping the infrastructure more efficient than running one isolated server per customer.

User-Specific Trading Engines

Each user receives a dedicated bot engine instance. This engine manages:

- User-specific Bybit API credentials
- User-specific trading configuration
- User-specific risk management
- User-specific Telegram notifications
- User-specific active positions
- User-specific trade history and state

This design allows different users to run different settings while still using the same backend infrastructure.

Market Data Architecture

Zion Core uses shared public WebSocket infrastructure for market data. This means that public Bybit market data does not need to be duplicated separately for every user.

This is important for scalability because multiple users can rely on the same public market stream while still maintaining separate private trading execution.

Private Execution Layer

Each active user engine has its own private Bybit WebSocket connection. This is required because order execution, positions, balances, and fills are private to each user account.

The bot monitors:

- Executions

- Position updates
- Stop-loss status
- Take-profit events
- Trailing-stop updates
- Trade reconciliation

Trading Pipeline

Zion Core follows a structured multi-phase trading pipeline:

1. Coin Monitoring
   The system scans eligible futures markets and adds coins for monitoring.
2. Dump Detection
   The bot identifies markets that have experienced a meaningful decline.
3. Stabilization
   The bot checks whether selling pressure has reduced and price has stopped aggressively declining.
4. Base Detection
   The system maps support and resistance zones to identify accumulation structures.
5. Armed Phase
   Once a valid base is detected, the bot waits for a breakout.
6. Breakout Confirmation
   The bot validates breakout strength using candle close, volume, body structure, spread, and slippage rules.
7. Trade Execution
   If all conditions pass, the bot calculates position size and places the trade.
8. Trade Management
   The system manages stop loss, TP1, breakeven protection, trailing stops, funding checks, and trade closure.

Risk Management

Zion Core includes built-in risk controls:

- Risk per trade
- Maximum concurrent trades
- Maximum total account risk
- Daily loss limit
- Margin usage limits
- Leverage limits
- Market regime filter

- Funding rate monitoring
- Circuit breaker protection

This makes the platform more than just a signal bot. It is a full execution and risk-management system.

Telegram Integration

The platform includes Telegram notifications for important bot events, including:

- Base detected
- Armed setup
- Entry
- TP1 hit
- Trailing activated
- Trade closed
- Funding warnings
- Market regime changes
- Errors and reconciliation issues

This allows users to monitor bot activity without constantly opening the dashboard.

Persistence and Recovery

The system stores coin contexts, open trades, engine state, and trade history in the database. On restart, Zion Core can restore previous state and reconnect to live Bybit positions.

This is important because the bot is designed to run continuously, and it needs to recover safely after server restarts or Railway redeployments.

Railway Deployment

The backend is deployed through Docker on Railway. Railway builds the backend from the Dockerfile, runs the FastAPI application through Uvicorn, and checks application health through the `/health` endpoint.

The architecture is suitable for early-stage SaaS deployment because one backend can support multiple active user engines.

Scalability Notes

The current architecture is good for the early version of Zion Core. The most efficient part is that public market WebSocket data is shared across engines.

The main future scalability concern is that each user engine may still perform its own scanning, candle processing, and pattern detection. If Zion Core grows to many users, the next optimization would be to separate shared market analysis from user-specific execution.

A future scalable model could be:

Shared Layer:

- Market scanning
- Candle fetching
- Pattern detection
- Base detection
- Breakout signals

User-Specific Layer:

- API credentials
- Risk settings
- Position sizing
- Trade execution
- Notifications
- Trade management

This would reduce duplicated work and make the platform cheaper to operate at scale.

Overall Assessment

The architecture is solid for an early-stage automated trading SaaS platform.

Strengths:

- Centralized Railway backend
- User-specific bot engines
- Shared public WebSocket market data
- Private user execution layer
- Persistent database recovery
- Full trade lifecycle management
- Integrated Telegram notifications
- Strong risk-management design

Main improvement area:

The platform should continue improving Pattern 1 logic, especially base detection and resistance-selection behavior. Later, if user count grows, shared signal generation should be separated from user-specific trade execution to improve scalability.