Phase 1: Local Docker Deployment Guide

Smart Money Concepts Trading Bot

Complete step-by-step guide to deploy the SMC trading bot locally using Docker containers.

Prerequisites

Before starting, ensure you have the following installed:

- **Docker Desktop** (Windows/Mac) or **Docker Engine** (Linux) - version 20.10+
- **Docker Compose** - version 2.0+
- **Git** - for version control
- **Text Editor** - VS Code, Sublime, or similar
- **Delta Exchange Testnet Account** - Sign up at https://testnet.delta.exchange

System Requirements

- **RAM:** Minimum 4GB (8GB recommended)
- **Storage:** Minimum 10GB free space
- **OS:** Windows 10/11, macOS 10.15+, or Linux (Ubuntu 20.04+)

Step 1: Project Setup

1.1 Create Project Directory

```
# Create main project folder
mkdir smc-trading-bot
cd smc-trading-bot

# Create subdirectories
mkdir -p backend frontend models deployment .github/workflows
mkdir -p frontend/static/css frontend/static/js
```

1.2 Initialize Git Repository

```
git init
git branch -M main
```

Step 2: Backend Files

2.1 Create `backend/requirements.txt`

```
Flask==3.0.0
Flask-CORS==4.0.0
requests==2.31.0
websocket-client==1.6.4
pandas==2.1.0
numpy==1.26.0
scikit-learn==1.3.2
tensorflow==2.15.0
python-dotenv==1.0.0
PyYAML==6.0.1
schedule==1.2.0
delta-rest-client==1.1.0
```

2.2 Create `backend/config.py`

```
import os
from dotenv import load_dotenv

load_dotenv()

# Environment mode: 'testnet' or 'live'
ENVIRONMENT_MODE = os.getenv('ENVIRONMENT_MODE', 'testnet')

# API Configuration
CONFIG = {
    'testnet': {
        'delta_api_url': 'https://testnet-api.delta.exchange',
        'delta_ws_url': 'wss://testnet-socket.delta.exchange',
        'api_key': os.getenv('DELTA_TESTNET_API_KEY', ''),
        'api_secret': os.getenv('DELTA_TESTNET_SECRET', ''),
    },
    'live': {
        'delta_api_url': 'https://api.delta.exchange',
        'delta_ws_url': 'wss://socket.delta.exchange',
        'api_key': os.getenv('DELTA_LIVE_API_KEY', ''),
        'api_secret': os.getenv('DELTA_LIVE_SECRET', ''),
    }
}

# Risk Management Parameters
RISK_PARAMS = {
    'max_risk_per_trade': 0.01,        # 1% risk per trade
    'max_position_size': 0.05,         # 5% max of account
    'reward_risk_ratio': 2.0,          # 2:1 reward:risk
    'max_drawdown': 0.15,              # 15% max drawdown
    'max_open_trades': 3,              # Max concurrent trades
}

# Trading Parameters
```

```
TRADING_PARAMS = {
    'symbols': ['BTCUSD', 'ETHUSD'],   # Trading pairs
    'timeframe': '15m',                 # Candle timeframe
    'order_block_lookback': 50,         # Bars to scan for OBs
    'choch_sensitivity': 0.002,         # 0.2% for CHoCH detection
    'min_engulfing_ratio': 1.5,         # Engulfing body ratio
}

def get_config():
    """Get current environment configuration"""
    return CONFIG[ENVIRONMENT_MODE]

def update_environment(env):
    """Switch between testnet and live"""
    global ENVIRONMENT_MODE
    if env in ['testnet', 'live']:
        ENVIRONMENT_MODE = env
        return True
    return False
```

2.3 Create `backend/delta_exchange_client.py`

```
import requests
import websocket
import json
import hmac
import hashlib
import time
from datetime import datetime
import threading
from config import get_config

class DeltaExchangeClient:
    """Client for Delta Exchange API integration"""

    def __init__(self):
        self.config = get_config()
        self.base_url = self.config['delta_api_url']
        self.ws_url = self.config['delta_ws_url']
        self.api_key = self.config['api_key']
        self.api_secret = self.config['api_secret']
        self.session = requests.Session()
        self.ws = None
        self.ws_connected = False

    def _generate_signature(self, method, endpoint, query_string='', payload=''):
        """Generate request signature"""
        timestamp = str(int(time.time()))
        signature_data = method + timestamp + endpoint + query_string + payload

        signature = hmac.new(
            self.api_secret.encode('utf-8'),
            signature_data.encode('utf-8'),
            hashlib.sha256
        ).hexdigest()
```

```
        return signature, timestamp

    def _get_headers(self, method, endpoint, query_string='', payload=''):
        """Generate request headers with signature"""
        signature, timestamp = self._generate_signature(
            method, endpoint, query_string, payload
        )

        return {
            'api-key': self.api_key,
            'timestamp': timestamp,
            'signature': signature,
            'User-Agent': 'smc-trading-bot/1.0',
            'Content-Type': 'application/json'
        }

    def get_ticker(self, symbol):
        """Get current ticker data for symbol"""
        endpoint = f'/v2/tickers/{symbol}'
        headers = self._get_headers('GET', endpoint)

        try:
            response = self.session.get(
                f"{self.base_url}{endpoint}",
                headers=headers,
                timeout=10
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Error fetching ticker: {e}")
            return None

    def fetch_candles(self, symbol, resolution='15', limit=500):
        """Fetch historical candle data"""
        endpoint = '/v2/history/candles'
        end_time = int(time.time())
        start_time = end_time - (limit * 60 * int(resolution.replace('m', '')))

        params = {
            'symbol': symbol,
            'resolution': resolution,
            'start': start_time,
            'end': end_time
        }

        query_string = '&amp;'.join([f"{k}={v}" for k, v in params.items()])
        headers = self._get_headers('GET', endpoint, f"?{query_string}")

        try:
            response = self.session.get(
                f"{self.base_url}{endpoint}",
                params=params,
                headers=headers,
                timeout=15
```

```
            )
            response.raise_for_status()
            data = response.json()

            if 'result' in data:
                return data['result']
            return None
        except Exception as e:
            print(f"Error fetching candles: {e}")
            return None

    def get_wallet_balance(self):
        """Get wallet balance"""
        endpoint = '/v2/wallet/balances'
        headers = self._get_headers('GET', endpoint)

        try:
            response = self.session.get(
                f"{self.base_url}{endpoint}",
                headers=headers,
                timeout=10
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Error fetching wallet balance: {e}")
            return None

    def place_order(self, product_id, side, size, order_type='market_order',
                    limit_price=None, stop_price=None):
        """Place a new order"""
        endpoint = '/v2/orders'

        payload = {
            'product_id': product_id,
            'size': size,
            'side': side,
            'order_type': order_type,
        }

        if order_type == 'limit_order' and limit_price:
            payload['limit_price'] = str(limit_price)

        if stop_price:
            payload['stop_order_type'] = 'stop_loss_order'
            payload['stop_price'] = str(stop_price)

        payload_str = json.dumps(payload)
        headers = self._get_headers('POST', endpoint, payload=payload_str)

        try:
            response = self.session.post(
                f"{self.base_url}{endpoint}",
                data=payload_str,
                headers=headers,
                timeout=10
```

```
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Error placing order: {e}")
            return None

    def cancel_order(self, product_id, order_id):
        """Cancel an existing order"""
        endpoint = f'/v2/orders'

        payload = {
            'product_id': product_id,
            'id': order_id
        }

        payload_str = json.dumps(payload)
        headers = self._get_headers('DELETE', endpoint, payload=payload_str)

        try:
            response = self.session.delete(
                f"{self.base_url}{endpoint}",
                data=payload_str,
                headers=headers,
                timeout=10
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Error canceling order: {e}")
            return None

    def get_positions(self):
        """Get all open positions"""
        endpoint = '/v2/positions'
        headers = self._get_headers('GET', endpoint)

        try:
            response = self.session.get(
                f"{self.base_url}{endpoint}",
                headers=headers,
                timeout=10
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Error fetching positions: {e}")
            return None

    def get_product_id(self, symbol):
        """Get product ID for a symbol"""
        ticker = self.get_ticker(symbol)
        if ticker and 'result' in ticker:
            return ticker['result'].get('product_id')
        return None
```

2.4 Create `backend/smc_strategy.py`

```
import pandas as pd
import numpy as np
from datetime import datetime

class SMCStrategy:
    """Smart Money Concepts trading strategy implementation"""

    def __init__(self, risk_params):
        self.risk_params = risk_params
        self.open_trades = []
        self.closed_trades = []

    def detect_order_blocks(self, df, lookback=50):
        """
        Detect order blocks (supply/demand zones)
        Order block = last candle before strong move in opposite direction
        """
        order_blocks = []

        if len(df) &lt; lookback + 2:
            return order_blocks

        df = df.reset_index(drop=True)

        for i in range(lookback, len(df) - 1):
            current_close = df.loc[i, 'close']
            current_open = df.loc[i, 'open']
            current_high = df.loc[i, 'high']
            current_low = df.loc[i, 'low']

            next_close = df.loc[i + 1, 'close']
            next_open = df.loc[i + 1, 'open']

            # Bullish order block: strong up candle followed by down move
            is_bullish_candle = current_close &gt; current_open
            is_strong_bullish = (current_close - current_open) &gt; (current_high - curre
            next_is_bearish = next_close &lt; next_open

            if is_bullish_candle and is_strong_bullish and next_is_bearish:
                order_blocks.append({
                    'type': 'BULLISH',
                    'high': current_high,
                    'low': current_low,
                    'open': current_open,
                    'close': current_close,
                    'index': i,
                    'timestamp': df.loc[i, 'time'] if 'time' in df.columns else i
                })

            # Bearish order block: strong down candle followed by up move
            is_bearish_candle = current_close &lt; current_open
            is_strong_bearish = (current_open - current_close) &gt; (current_high - curre
            next_is_bullish = next_close &gt; next_open
```

```
            if is_bearish_candle and is_strong_bearish and next_is_bullish:
                order_blocks.append({
                    'type': 'BEARISH',
                    'high': current_high,
                    'low': current_low,
                    'open': current_open,
                    'close': current_close,
                    'index': i,
                    'timestamp': df.loc[i, 'time'] if 'time' in df.columns else i
                })

        return order_blocks[-5:] if order_blocks else []  # Return last 5

    def detect_choch(self, df, sensitivity=0.002):
        """
        Detect Change of Character (market structure break)
        CHoCH = break of previous high/low indicating trend change
        """
        choch_signals = []

        if len(df) &lt; 20:
            return choch_signals

        df = df.reset_index(drop=True)

        for i in range(20, len(df)):
            # Calculate recent swing high and low
            recent_high = df.loc[i-20:i-1, 'high'].max()
            recent_low = df.loc[i-20:i-1, 'low'].min()

            current_high = df.loc[i, 'high']
            current_low = df.loc[i, 'low']

            # Bullish CHoCH: price breaks above recent high
            if current_high &gt; recent_high * (1 + sensitivity):
                choch_signals.append({
                    'type': 'BULLISH_CHOCH',
                    'price': current_high,
                    'previous_high': recent_high,
                    'index': i,
                    'timestamp': df.loc[i, 'time'] if 'time' in df.columns else i
                })

            # Bearish CHoCH: price breaks below recent low
            if current_low &lt; recent_low * (1 - sensitivity):
                choch_signals.append({
                    'type': 'BEARISH_CHOCH',
                    'price': current_low,
                    'previous_low': recent_low,
                    'index': i,
                    'timestamp': df.loc[i, 'time'] if 'time' in df.columns else i
                })

        return choch_signals[-3:] if choch_signals else []  # Return last 3

    def detect_engulfing(self, df, min_ratio=1.5):
```

```
        """
        Detect engulfing candlestick patterns
        Engulfing = current candle completely engulfs previous candle
        """
        engulfing_patterns = []

        if len(df) &lt; 2:
            return engulfing_patterns

        df = df.reset_index(drop=True)

        for i in range(1, len(df)):
            prev_open = df.loc[i-1, 'open']
            prev_close = df.loc[i-1, 'close']
            prev_body = abs(prev_close - prev_open)

            curr_open = df.loc[i, 'open']
            curr_close = df.loc[i, 'close']
            curr_body = abs(curr_close - curr_open)

            if prev_body == 0:
                continue

            body_ratio = curr_body / prev_body

            # Bullish engulfing
            is_prev_bearish = prev_close &lt; prev_open
            is_curr_bullish = curr_close &gt; curr_open
            curr_engulfs_prev = curr_close &gt; prev_open and curr_open &lt; prev_close

            if is_prev_bearish and is_curr_bullish and curr_engulfs_prev and body_ratio &
                engulfing_patterns.append({
                    'type': 'BULLISH_ENGULFING',
                    'index': i,
                    'strength': body_ratio,
                    'close': curr_close,
                    'timestamp': df.loc[i, 'time'] if 'time' in df.columns else i
                })

            # Bearish engulfing
            is_prev_bullish = prev_close &gt; prev_open
            is_curr_bearish = curr_close &lt; curr_open
            curr_engulfs_prev = curr_close &lt; prev_open and curr_open &gt; prev_close

            if is_prev_bullish and is_curr_bearish and curr_engulfs_prev and body_ratio &
                engulfing_patterns.append({
                    'type': 'BEARISH_ENGULFING',
                    'index': i,
                    'strength': body_ratio,
                    'close': curr_close,
                    'timestamp': df.loc[i, 'time'] if 'time' in df.columns else i
                })

        return engulfing_patterns[-2:] if engulfing_patterns else []  # Return last 2

    def generate_signal(self, df, symbol):
```

```
        """Generate trading signal based on SMC analysis"""
        if df is None or len(df) &lt; 50:
            return None

        # Detect SMC components
        order_blocks = self.detect_order_blocks(df)
        choch = self.detect_choch(df)
        engulfing = self.detect_engulfing(df)

        # Current price
        current_price = df.iloc[-1]['close']

        # Determine signal confidence
        bullish_signals = 0
        bearish_signals = 0

        # Check order blocks
        if order_blocks:
            latest_ob = order_blocks[-1]
            if latest_ob['type'] == 'BULLISH':
                bullish_signals += 1
            elif latest_ob['type'] == 'BEARISH':
                bearish_signals += 1

        # Check CHoCH
        if choch:
            latest_choch = choch[-1]
            if latest_choch['type'] == 'BULLISH_CHOCH':
                bullish_signals += 1
            elif latest_choch['type'] == 'BEARISH_CHOCH':
                bearish_signals += 1

        # Check engulfing
        if engulfing:
            latest_engulfing = engulfing[-1]
            if latest_engulfing['type'] == 'BULLISH_ENGULFING':
                bullish_signals += 2  # Engulfing has more weight
            elif latest_engulfing['type'] == 'BEARISH_ENGULFING':
                bearish_signals += 2

        # Calculate confidence (0 to 1)
        total_signals = bullish_signals + bearish_signals
        if total_signals == 0:
            confidence = 0.5
            direction = 'NEUTRAL'
        elif bullish_signals &gt; bearish_signals:
            confidence = bullish_signals / (total_signals + 3)  # Normalize
            direction = 'BUY'
        else:
            confidence = bearish_signals / (total_signals + 3)
            direction = 'SELL'

        return {
            'symbol': symbol,
            'direction': direction,
            'confidence': confidence,
```

```
            'current_price': current_price,
            'order_blocks': order_blocks,
            'choch': choch[-1] if choch else None,
            'engulfing': engulfing[-1] if engulfing else None,
            'timestamp': datetime.now().isoformat()
        }

    def calculate_position(self, signal, account_balance):
        """Calculate position size, entry, SL, and TP"""
        if not signal or signal['confidence'] &lt; 0.65:
            return None

        # Risk amount per trade
        risk_amount = account_balance * self.risk_params['max_risk_per_trade']

        current_price = signal['current_price']
        direction = signal['direction']

        # Use order blocks for SL placement
        if signal['order_blocks']:
            latest_ob = signal['order_blocks'][-1]

            if direction == 'BUY' and latest_ob['type'] == 'BULLISH':
                entry_price = current_price
                stop_loss = latest_ob['low'] * 0.999  # Slightly below OB low
                take_profit = entry_price + (entry_price - stop_loss) * self.risk_params[

            elif direction == 'SELL' and latest_ob['type'] == 'BEARISH':
                entry_price = current_price
                stop_loss = latest_ob['high'] * 1.001  # Slightly above OB high
                take_profit = entry_price - (stop_loss - entry_price) * self.risk_params[
            else:
                return None
        else:
            return None

        # Calculate position size
        sl_distance = abs(entry_price - stop_loss)
        if sl_distance == 0:
            return None

        position_size = risk_amount / sl_distance

        # Limit position size
        max_position = account_balance * self.risk_params['max_position_size']
        position_size = min(position_size, max_position / entry_price)

        return {
            'entry_price': entry_price,
            'stop_loss': stop_loss,
            'take_profit': take_profit,
            'position_size': round(position_size, 4),
            'side': 'buy' if direction == 'BUY' else 'sell',
            'risk_amount': risk_amount,
            'potential_profit': abs(take_profit - entry_price) * position_size
        }
```

Step 3: Continue with remaining files...

Due to length constraints, the complete guide continues with:

- ML Model implementation
- Flask API server
- Frontend dashboard integration
- Docker configurations
- Environment setup
- Testing procedures
- Troubleshooting guide

[1] [71][72][^75]

[2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20]

⁂

1. https://docs.delta.exchange
2. https://help.ovhcloud.com/csm/en-vps-deploy-website-github-actions?id=kb_article_view&sysparm_article=KB0066184
3. https://deltaexchangeindia.freshdesk.com/support/solutions/articles/80001181136-getting-started-with-delta-exchange-api-copilot
4. https://docs.docker.com/reference/compose-file/services/
5. https://www.hostinger.com/support/deploy-to-hostinger-vps-using-github-actions/
6. https://www.youtube.com/watch?v=TSzz-Fllq14
7. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/multi-container-applications-docker-compose
8. https://blog.ando.ai/posts/github-actions-vps-deployment/
9. https://www.youtube.com/watch?v=O5i5DJUAap4
10. https://docs.docker.com/compose/how-tos/multiple-compose-files/
11. https://docs.github.com/actions/deployment/about-deployments/deploying-with-github-actions
12. https://github.com/delta-exchange/python-rest-client
13. https://www.delta.exchange/support/solutions/articles/80001153884-your-complete-api-faq-guide-everything-you-need-to-know
14. https://www.delta.exchange/algo/delta-exchange-apis
15. https://www.delta.exchange/blog/introduction-to-algorithmic-trading-with-delta-rest-client
16. https://www.profitaddaweb.com/2025/04/delta-exchange-api-in-python.html
17. https://stackoverflow.com/questions/49191304/how-to-deal-with-multiple-services-inside-a-docker-compose-yml
18. https://gist.github.com/danielwetan/4f4db933531db5dd1af2e69ec8d54d8a
19. https://www.youtube.com/watch?v=RoVeASkeMpc

20. https://matthiasnoback.nl/2018/03/defining-multiple-similar-services-with-docker-compose/