Selling Options — Environments Runbook (Per-Env, IPs Included)

LAB (One-Page Tester) • DEV on NAS (192.168.217.150) • PROD on EC2 (selling-options.com)

# Table of Contents

# LAB (One-Page Tester on NAS)

Environment host: 192.168.217.150

Purpose: Safe sandbox to test a single page + API without touching DEV/PROD (same-origin via Nginx).

## File Structure

/volume1/docker/selling-options-lab  
├─ compose.yaml  
├─ nginx.conf  
├─ api/  
│ ├─ Dockerfile  
│ ├─ requirements.txt  
│ └─ options\_results\_api.py  
└─ static/  
 └─ index.html

## Services & Ports

- lab-static (Nginx): publishes 5080→80; serves `/` and proxies `/api/\*` to `lab-api:5000`.

- lab-api (Flask+yfinance): internal port 5000 only.

## Operate

cd /volume1/docker/selling-options-lab  
sudo docker compose up -d --build  
sudo docker compose ps  
sudo docker compose logs -f lab-api  
sudo docker compose restart lab-static # after nginx.conf change

## Open & Test

http://192.168.217.150:5080/  
curl -i "http://192.168.217.150:5080/api/get\_options\_data?symbol=NVDA"  
curl -i "http://192.168.217.150:5080/api/results\_both?symbol=NVDA&date=2025-08-29"

## Credentials (Placeholders)

LAB\_APP\_USER=replace\_me  
LAB\_APP\_PASS=replace\_me  
LAB\_API\_TOKEN=replace\_me

## Troubleshooting

• 404 on `/api/\*`: ensure `proxy\_pass http://lab-api:5000` (NO trailing slash).

• Port conflict: change published port (e.g., 5081:80).

# DEV (NAS)

Environment host: 192.168.217.150

Purpose: Integration staging for the full site on the NAS.

## File Structure

/volume1/docker/selling-options-app  
├─ Dockerfile  
├─ Caddyfile # reverse proxy for PROD on NAS (if used)  
├─ compose.yaml # app + caddy (for PROD-on-NAS scenario)  
├─ main.py # Flask entry (served by Gunicorn)  
├─ templates/ # e.g., price\_calculator.html  
├─ static/ # CSS/JS/images  
├─ pc\_requirements.txt # Python dependencies  
└─ other modules # e.g., bullvsbear.py, optionsflow.py, etc.

## Services & Ports

- selling-options-dev: publishes 5050→5000 (Gunicorn listens on 5000).

## Database (Synology MariaDB Package)

Synology \*\*MariaDB10\*\* is installed as a package (not Docker).

Based on config (`/var/packages/MariaDB10/etc/my\_port.cnf`), the active port appears to be \*\*3306\*\*.

Verify the port is actually listening and reachable:

# On the NAS (SSH)  
ss -tulpen | egrep '3306|3307' || netstat -tulpen | egrep '3306|3307'  
  
# Test login (adjust port if needed)  
mysql -h 192.168.217.150 -P 3306 -u root -p

### Create DB & Users (DEV)

Run these once in the MySQL shell after connecting as `root`. Replace placeholders; do not commit secrets.

CREATE DATABASE selling\_options CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;  
  
-- Primary app user (read/write)  
CREATE USER 'so\_dev'@'%' IDENTIFIED BY 'REPLACE\_DEV\_DB\_PASS';  
GRANT ALL PRIVILEGES ON selling\_options.\* TO 'so\_dev'@'%';  
  
-- Optional read-only user (for helpers/BI)  
CREATE USER 'so\_read'@'%' IDENTIFIED BY 'REPLACE\_DEV\_DB\_RO\_PASS';  
GRANT SELECT ON selling\_options.\* TO 'so\_read'@'%';  
  
FLUSH PRIVILEGES;

### App → DB Connection (DEV)

Store credentials in `.env` (do NOT commit):

DEV\_DB\_HOST=192.168.217.150  
DEV\_DB\_PORT=3306 # update if your check shows 3307  
DEV\_DB\_NAME=selling\_options  
DEV\_DB\_USER=so\_dev  
DEV\_DB\_PASS=REPLACE\_DEV\_DB\_PASS

SQLAlchemy URL example:

mysql+pymysql://so\_dev:${DEV\_DB\_PASS}@${DEV\_DB\_HOST}:${DEV\_DB\_PORT}/${DEV\_DB\_NAME}?charset=utf8mb4

### Indexing & Query Tips

USE selling\_options;  
-- Helpful compound index for common lookups:  
CREATE INDEX idx\_import\_symbol\_exp ON Import (Symbol, `Exp Date`);  
EXPLAIN SELECT \* FROM Import WHERE Symbol='NVDA' AND `Exp Date`='2025-08-29';

## Operate DEV App

cd /volume1/docker/selling-options-app  
sudo docker build -t selling-options:dev -f Dockerfile .  
sudo docker rm -f selling-options-dev 2>/dev/null || true  
sudo docker run -d --name selling-options-dev -p 5050:5000 --env-file .env selling-options:dev  
sudo docker logs -f selling-options-dev  
Open DEV: http://192.168.217.150:5050/price-calculator

## Credentials (Placeholders for DEV)

# App auth (if used)  
DEV\_APP\_USER=replace\_me  
DEV\_APP\_PASS=replace\_me  
  
# MariaDB (Synology Package)  
DEV\_DB\_HOST=192.168.217.150  
DEV\_DB\_PORT=3306  
DEV\_DB\_NAME=selling\_options  
DEV\_DB\_USER=so\_dev  
DEV\_DB\_PASS=REPLACE\_DEV\_DB\_PASS  
  
# Optional read-only user  
DEV\_DB\_RO\_USER=so\_read  
DEV\_DB\_RO\_PASS=REPLACE\_DEV\_DB\_RO\_PASS

## Troubleshooting (DEV)

• Connection refused: confirm port (3306 vs 3307), firewall rules, and that the MariaDB package is running.

• Auth errors: verify user grants; re-run `GRANT` and `FLUSH PRIVILEGES`.

• From Docker to NAS DB: use the NAS LAN IP; `host.docker.internal` doesn’t exist on Synology.

# PROD (EC2)

Public domain: selling-options.com

Public IP (record this for ops): <PUBLIC\_IP\_OF\_selling-options.com> ← Fill after DNS lookup or cloud console

## File Structure

/opt/selling-options-app  
├─ Dockerfile  
├─ compose.yaml # app + reverse proxy  
├─ main.py  
├─ templates/  
├─ static/  
├─ requirements.txt (or pc\_requirements.txt)  
└─ .env # not committed (holds secrets)

## Services & Ports

- proxy (Caddy or Nginx): publishes 80/443 to the internet.

- app: Flask/Gunicorn on internal port 5000.

- database: RDS MariaDB/Aurora MySQL recommended; allow only the app SG to connect.

## Database (PROD plan)

# SQLAlchemy (example)  
mysql+pymysql://so\_prod:${PROD\_DB\_PASS}@${PROD\_DB\_HOST}:3306/${PROD\_DB\_NAME}?charset=utf8mb4

## Operate (PROD)

cd /opt/selling-options-app  
sudo docker compose up -d --build  
sudo docker compose ps  
sudo docker compose logs -f app  
sudo docker compose logs -f proxy  
curl -I http://selling-options.com/price-calculator

## Credentials (Placeholders for PROD)

PROD\_DOMAIN=selling-options.com  
PROD\_PUBLIC\_IP=<PUBLIC\_IP\_OF\_selling-options.com>  
  
PROD\_APP\_USER=replace\_me  
PROD\_APP\_PASS=replace\_me  
  
PROD\_DB\_HOST=<rds-endpoint or internal host>  
PROD\_DB\_PORT=3306  
PROD\_DB\_NAME=selling\_options  
PROD\_DB\_USER=so\_prod  
PROD\_DB\_PASS=REPLACE\_PROD\_DB\_PASS

## Backups & Snapshots (PROD)

• Use RDS automated backups and snapshots for DB.

• Use AMIs and EBS snapshots for EC2.

## Troubleshooting (PROD)

• 502/Bad Gateway: proxy can’t reach `app:5000` → check container names and `depends\_on`.

• TLS issues: confirm DNS points to the EC2 instance and security group allows 80/443.

• DB auth errors: confirm security groups and credentials.

# Summary

### Overall Purpose

It’s a full-stack web application called Selling Options that predicts a stock’s future price using options data (calls and puts).

The app pulls live options chains from Yahoo Finance (yfinance), applies your custom weighted average math (similar to your Access database logic), and produces a predicted stock price vs current price.

Users interact with a web UI where they type in a ticker (like NVDA), select an expiration date, and then see the computed prediction.

### Frontend (UI)

index.html is a sleek single-page app:

Input box for stock symbol.

Dropdown for expiration dates (auto-loaded from API).

Button to “Get Prediction.”

Displays current price, predicted price, and % change.

All interactions are JavaScript → API calls (/api/\* endpoints).

### Backend (API & Logic)

options\_results\_api.py (Flask app):

/api/get\_options\_data: fetches expiration dates or full option chains.

/api/results\_both: computes detailed option analytics (like your Access queries).

/api/prediction: simplified endpoint for UI — returns current price, predicted price, and % change.

Uses pandas, numpy, yfinance for fetching & calculating.

### Math / Prediction Logic

For each option:

BreakEven = Strike + LastPrice

TotPre = LastPrice × Open Interest

PercentofMoneySpent = TotPre / SumOfTotPre

PartofMoney = BreakEven × PercentofMoneySpent

Final Predicted Price = Σ PartofMoney (weighted average).

### Infrastructure / Deployment

Defined in your Runbook:

LAB: sandbox, one-page tester (runs on NAS at 192.168.217.150:5080).

DEV: full app staging on NAS (selling-options-dev, MariaDB backend).

PROD: live site at selling-options.com on AWS EC2 with Docker + Caddy/Nginx.

Uses Docker (compose.yaml, Dockerfile) to containerize API + frontend + proxy.

### Other Tech

nginx.conf: serves static UI + proxies /api/\* to Flask container.

requirements.txt: Flask, pandas, yfinance, numpy.

MariaDB: planned for DEV/PROD storage (Synology package in DEV, RDS in PROD).

✅ In short:  
This is your options-trading prediction web app.

Frontend: clean UI where users enter a stock + expiration.

Backend: Flask API with your custom prediction math.

Deployment: Dockerized, running across LAB/DEV/PROD environments (NAS + AWS).

Goal: Give users a simple “predicted price vs current” insight based on options open interest weighting.

Do you want me to also map out how the pieces talk to each other (flow diagram) so you have a visual of request → API → prediction → frontend?

##### You said:

yes

###### ChatGPT said:

Thought for 19s

Awesome—here’s a clear, at-a-glance flow of how everything talks to everything else.

# High-Level Flow (All Environments)

User (Browser)

│ enters symbol → picks expiration → clicks "Get Prediction"

▼

Web UI (index.html)

│ fetches /api/get\_options\_data and /api/prediction

▼

Reverse Proxy (Nginx/Caddy)

│ static / → index.html

│ /api/\* → Flask app (same origin)

▼

Flask API (options\_results\_api.py)

│ \_fetch\_expirations / \_fetch\_chain via yfinance

│ \_compute\_results (weighted BreakEven math across Calls+Puts)

│ \_get\_current\_price via yfinance

▼

Response JSON

│ { currentPrice, predictedPrice, pctChange, … }

▼

Web UI renders tiles (current vs predicted, % change)

# LAB (One-Page Tester on NAS)

User → http://192.168.217.150:5080/

|

| (Static + Proxy)

v

Nginx (lab-static, port 80 published as 5080)

| / → serves index.html

| /api/\* → proxy\_pass http://lab-api:5000

v

Flask (lab-api:5000 in container)

|

| yfinance: options & prices

v

Compute prediction → JSON → back through Nginx → Browser

Key endpoints:

/api/get\_options\_data?symbol=NVDA → returns expirations (or full chain if date provided)

/api/prediction?symbol=NVDA&date=YYYY-MM-DD → returns current & predicted price

# DEV (Full App on NAS)

User → http://192.168.217.150:5050/price-calculator

|

| (Reverse proxy optional; often direct port map 5050→5000)

v

Gunicorn/Flask app (selling-options-dev, port 5000)

|\

| \\_\_ yfinance (live options + prices)

|

└── MariaDB (Synology package, likely :3306)

- stores/imports your options data as needed

Notes:

DB connection via mysql+pymysql://...:3306/...

Helpful index for lookups: (Symbol, Exp Date)

# PROD (EC2, selling-options.com)

User → https://selling-options.com

|

| 443/80 (TLS & static)

v

Caddy/Nginx (proxy)

| / → static (if any)

| /api/\* → app:5000

v

Gunicorn/Flask app (container)

|\

| \\_\_ yfinance (options + prices)

|

└── RDS (MariaDB/Aurora MySQL, :3306)

- managed backups/snapshots

# Request Lifecycles (Typical)

Load expirations (auto on typing symbol):

UI → GET /api/get\_options\_data?symbol=SYM

API → yfinance Ticker(SYM).options → returns [dates...]

UI populates the dropdown.

Run prediction:

UI → GET /api/prediction?symbol=SYM&date=YYYY-MM-DD

API → yfinance option\_chain(date) → calls+puts DataFrames

Compute:

BreakEven = Strike + LastPrice

TotPre = LastPrice \* OI

weight = TotPre / ΣTotPre

PartofMoney = BreakEven \* weight

Predicted = Σ PartofMoney

API → yfinance for current price

Return JSON { currentPrice, predictedPrice, pctChange }

UI renders tiles.

# Useful Commands

Print all the files in a directory and its subdirectories.

find . -print | grep -v '^\.$' | sed 's|^\./||' | sort | awk -F/ '{for(i=1;i<NF;i++) printf "│ "; print "├── " $NF}'

# Step B — v0903 Updates (NAS)

1. DSM Reverse Proxy + Caddy (instead of nginx in LAB):  
 - LAB external: https://selling-options-lab.com → DSM → Caddy → app:5000  
 - DEV external: https://selling-options-dev.com → DSM → Caddy → app:5000  
  
2. Ports (updated):  
 - DEV: 443 → 5081:80 → app:5000  
 - LAB: 443 → 5082:80 → app:5000  
 - DEV Caddy sets header `X-Env: DEV`, LAB does not.  
  
3. Sync Scripts:  
 - /volume1/docker/lab-sync.sh → DEV → LAB sync  
 - /volume1/docker/dev-sync-safe.sh → LAB → DEV sync (dry-run by default, --apply to execute)  
  
Examples:  
 bash dev-sync-safe.sh # DRY RUN  
 bash dev-sync-safe.sh --apply # Apply LAB → DEV code only  
 bash dev-sync-safe.sh --apply --include-config --force-proxy-fix  
  
4. Troubleshooting (extra from v0903):  
 - `failed to set GOMAXPROCS` in caddy logs: benign on Synology  
 - If hostnames don’t route: DSM Control Panel → Login Portal → Reverse Proxy

# Math / Prediction Logic (Authoritative Spec)

This section replaces earlier math definitions. All prediction logic follows the \*\*Prediction Math Specification\*\* (2025-09) exactly.  
  
\*\*Prediction Math Specification — Selling-Options (Volume/OI Weighting)\*\*  
  
\*\*Purpose:\*\* Compute predicted stock price from an options chain for a single underlying and expiration date.  
  
\*\*Filters:\*\* lastPrice > 0 AND (Volume > 0 in volume mode | OI > 0 in OI mode).  
  
\*\*Breakeven (BE):\*\*  
- Calls: BE = strike + premium  
- Puts: BE = strike − premium  
  
\*\*Weights:\*\*  
- Volume weight: premium × volume  
- OI weight: premium × OI  
  
\*\*Predictions:\*\*  
- P\_vol = Σ(BEᵢ × wᵢ(vol)) / Σ wᵢ(vol)  
- P\_oi = Σ(BEᵢ × wᵢ(oi)) / Σ wᵢ(oi)  
- P\_avg = (P\_vol + P\_oi) / 2  
- P\_blend(α) = α·P\_oi + (1−α)·P\_vol  
  
\*\*Derived Metric:\*\* pctChange = (P(mode) − CurrentPrice) / CurrentPrice  
  
\*\*Debug:\*\* include sum of weights + row count  
  
\*\*Worked Example (NVDA, 2025-09-05):\*\*  
- Σ w\_vol = 2,162,826.65 across 141 rows  
- P\_vol = 169.464602

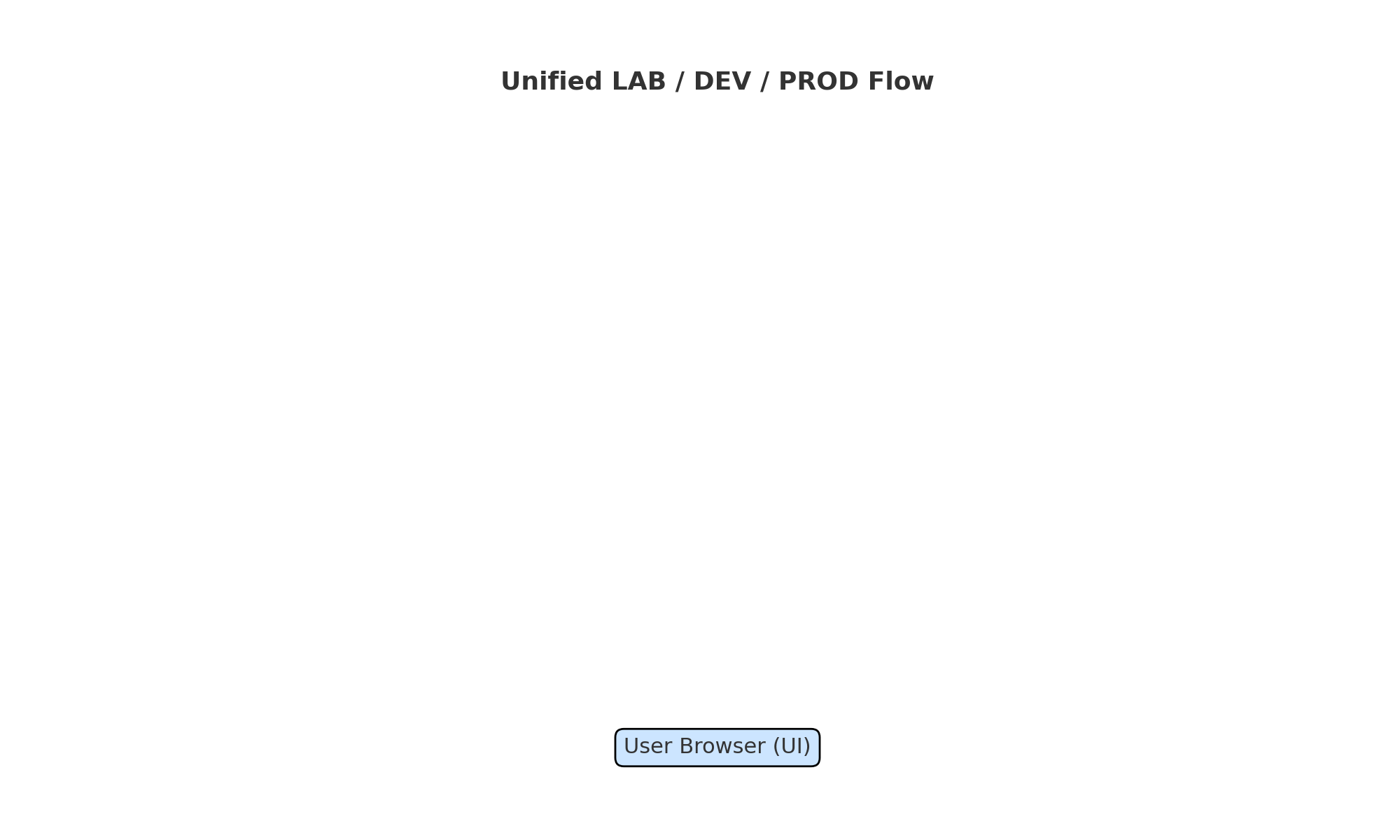
# PROD (AWS EC2 — Verified State 2025-09)

\*\*Host & System Info:\*\*  
- Instance: t3.micro (AWS EC2)  
- OS: Ubuntu 24.04.3 LTS  
- Kernel: 6.14.0-1011-aws  
- Public IP: 98.86.15.115  
- Domain: https://selling-options.com  
- Disk: 8 GB root (76% used, ~1.7 GB free)  
- RAM: 914 MiB, no swap configured  
  
\*\*Docker Containers (live):\*\*  
- caddy:2 → ports 80/443, Restart=always  
- selling-options:latest → Gunicorn app on :5000, Restart=always  
  
\*\*Caddyfile (live inside container):\*\*  
selling-options.com {  
 encode zstd gzip  
 header {  
 Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"  
 X-Content-Type-Options "nosniff"  
 X-Frame-Options "DENY"  
 Referrer-Policy "no-referrer-when-downgrade"  
 }  
 reverse\_proxy selling-options:5000  
}  
  
\*\*TLS & Certificates:\*\*  
- Cached inside container at /data/caddy (persisted via Docker volume)  
  
\*\*Health Checks:\*\*  
- curl -I https://selling-options.com → HTTP/2 200 OK  
- /api/get\_options\_data?symbol=NVDA → expirations OK  
- /api/prediction?symbol=NVDA&date=2025-09-05 → 404 Not Found  
  
\*\*Issues Observed:\*\*  
- Low disk space on root volume  
- No swap file (OOM risk on t3.micro)  
- Prediction endpoint missing or outdated in current image

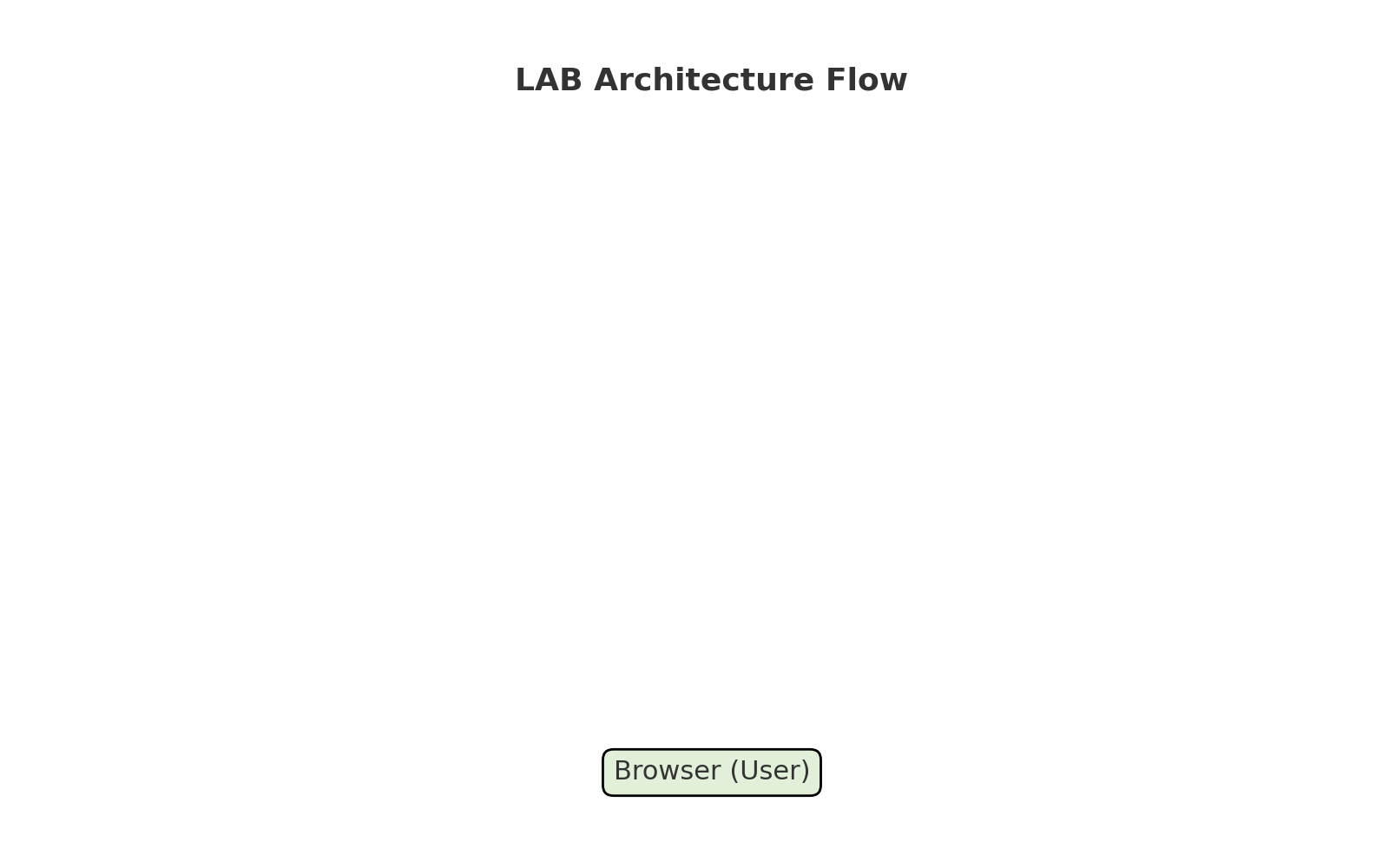
# Architecture Diagrams

The following diagrams illustrate Unified, LAB, DEV, and PROD flows.

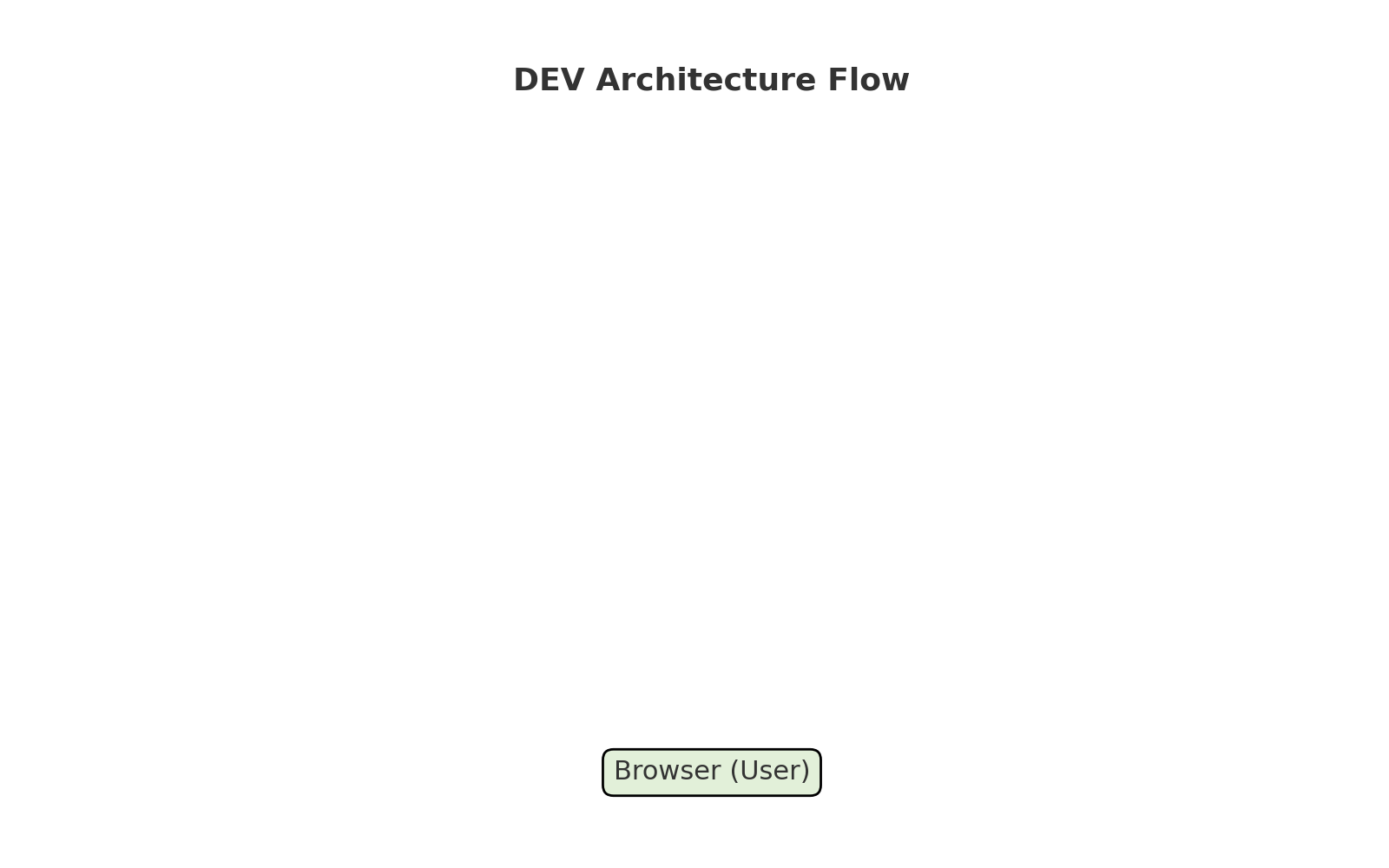
## Unified Overview Flow



## LAB Flow



## DEV Flow



## PROD Flow

