Contents

VPS Infrastructure Plan: Backtesting & Development VPS

Comprehensive Setup Guide for Optimal Development & Testing Environment

Date: January 2025

Version: 1.0

Status: Implementation Plan

Executive Summary

This document provides a comprehensive plan for implementing Backtesting VPS and Development VPS infrastructure to optimize the Argo-Alpine trading platform development and testing workflows. The plan includes detailed setup instructions, integration strategies, cost-benefit analysis, and optimal use cases.

Recommendation: YES - Implement Both VPS Servers

- **Backtesting VPS**: High Priority (Week 1)
- **Dev VPS**: Medium Priority (Month 2)
- **Total Investment**: $45-55/month
- **Expected ROI**: 20-30 hours/month saved, 3-5x faster backtesting

---

Table of Contents

Current Architecture Analysis

Current Infrastructure

```
                    CURRENT ARCHITECTURE



  MacBook (Local Development)
     Code editing
     Quick testing
     Light backtesting (slow, limited parallel)
     Git workflow

  Production Argo (178.156.194.174)
     Signal generation (24/7, every 5 seconds)
     Live trading execution
     Real-time data processing
     Redis caching
     PostgreSQL (historical data)

  Production Alpine (91.98.153.49)
     User-facing API
     Frontend (Next.js)
     PostgreSQL (user data)
     Redis (sessions)
```

Current Limitations

Component	Limitation	Impact
MacBook backtesting	Slow (5-10 min per symbol)	Limited iteration speed
MacBook parallel processing	Limited (thermal throttling)	Can't test many symbols simultaneously
MacBook overnight processing	Can't run (sleep issues)	No batch processing capability
Dev environment	MacBook only	No 24/7 testing environment
Integration testing	Manual, slow	Limited test coverage

Current Workflow Challenges

1. **Backtesting Bottleneck**
   - Single symbol, 5 years: 5-10 minutes
   - 10 symbols, 5 years: 50-100 minutes (sequential)
   - Parallel processing limited by MacBook thermal constraints
   - Overnight batch processing impossible
2. **Development Limitations**
   - No 24/7 dev environment
   - Long-running tests difficult
   - No automated CI/CD
   - Integration testing manual
3. **Resource Constraints**
   - MacBook tied up during heavy backtesting
   - Can't run multiple heavy processes simultaneously
   - Battery/thermal limitations

Proposed Architecture

Optimized Infrastructure

```
                  OPTIMIZED ARCHITECTURE


  MacBook (Local Development)
     Code editing (best experience)
     Quick testing (fast iteration)
     Light backtesting (single symbol, short periods)
     Git workflow

  Dev VPS ($15-20/mo)
     Full dev stack 24/7
     Integration testing
     CI/CD pipeline
     Long-running tests
     Shared dev environment

  Backtesting VPS ($30-40/mo)
     Heavy backtesting (parallel)
     Multi-year analysis
     Strategy optimization
     Overnight batch processing
     Walk-forward analysis

  Production Argo (178.156.194.174)
     Signal generation (24/7, every 5 seconds)
     Live trading execution
     Real-time data processing
     Redis + PostgreSQL

  Production Alpine (91.98.153.49)
```

```
User-facing API
Frontend
PostgreSQL + Redis
```

Key Improvements

1. **Separation of Concerns**
   - MacBook: Code editing and quick testing
   - Dev VPS: 24/7 development and integration testing
   - Backtesting VPS: Heavy computational work
   - Production: Live trading and user-facing services
2. **Performance Gains**
   - 3-5x faster backtesting
   - Parallel processing enabled
   - Overnight batch processing
   - 24/7 automated testing
3. **Workflow Optimization**
   - Faster iteration cycles
   - Automated CI/CD
   - Better resource utilization
   - Scalable infrastructure

Phase 1: Backtesting VPS Setup

1.1 VPS Specifications

Recommended Provider: Hetzner (Best Price/Performance)

Specifications: - **CPU**: 8 cores (AMD EPYC or Intel Xeon) - **RAM**: 16GB - **Storage**: 200GB NVMe SSD - **Network**: 1Gbps - **Location**: US East (low latency to data sources) - **Cost**: ~$30-35/month

Alternative Providers: - **DigitalOcean**: 8GB/4vCPU - $48/month (easier setup) - **Linode**: 8GB/4vCPU - $40/month (good balance)

1.2 Initial Server Setup

```
# 1. Provision VPS
#    - Hetzner: CPX41 (8 cores, 16GB, 200GB) - $30/month
#    - Ubuntu 22.04 LTS
#    - SSH key authentication

# 2. Initial server setup
ssh root@backtest-vps-ip

# Update system
apt update && apt upgrade -y

# Install base tools
apt install -y git python3.11 python3.11-venv python3-pip \
    build-essential libpq-dev redis-server postgresql-client \
    htop iotop nginx certbot

# Create user
adduser argo
usermod -aG sudo argo
mkdir -p /home/argo/workspace
chown argo:argo /home/argo/workspace
```

1.3 Environment Setup

```
# Switch to argo user
su - argo
cd /home/argo/workspace

# Clone repository
git clone <your-repo-url> argo-alpine-workspace
cd argo-alpine-workspace

# Setup Python environment
```

```
cd argo
python3.11 -m venv venv
source venv/bin/activate
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt

# Install additional backtesting dependencies
pip install jupyter notebook pandas-profiling plotly
```

1.4 Configuration

```
# Create backtesting-specific config
cat > /home/argo/workspace/argo-alpine-workspace/argo/.env.backtest << 'EOF'
# Backtesting VPS Configuration
ENVIRONMENT=backtesting
ARGO_24_7_MODE=false  # Not needed for backtesting
ALPACA_API_KEY_ID=${ALPACA_API_KEY_ID}  # From secrets
ALPACA_SECRET_KEY=${ALPACA_SECRET_KEY}  # From secrets
ALPACA_PAPER=true
DATABASE_PATH=/home/argo/workspace/backtest_data/signals.db
REDIS_HOST=localhost
REDIS_PORT=6379
LOG_LEVEL=INFO
BACKTEST_MODE=true
PARALLEL_WORKERS=8  # Match CPU cores
EOF

# Create data directory
mkdir -p /home/argo/workspace/backtest_data
mkdir -p /home/argo/workspace/backtest_results
```

1.5 Backtesting Scripts Setup

Create optimized parallel backtesting script:

```python
#!/usr/bin/env python3
"""
Parallel batch backtesting script for VPS
Optimized for multi-core processing
"""
import asyncio
import sys
from pathlib import Path
sys.path.insert(0, str(Path(__file__).parent.parent.parent / "argo"))

from argo.backtest.strategy_backtester import StrategyBacktester
from argo.backtest.constants import BacktestConstants
from datetime import datetime, timedelta
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

async def backtest_symbol(symbol: str, years: int = 5):
    """Backtest single symbol"""
    try:
        backtester = StrategyBacktester(
            initial_capital=BacktestConstants.DEFAULT_INITIAL_CAPITAL,
            use_cost_modeling=True,
            use_enhanced_cost_model=True
        )

        end_date = datetime.now()
        start_date = end_date - timedelta(days=years*365)

        result = await backtester.run_backtest(
            symbol,
            start_date=start_date,
            end_date=end_date,
            min_confidence=BacktestConstants.DEFAULT_MIN_CONFIDENCE
        )

        return symbol, result
    except Exception as e:
        logger.error(f"Error backtesting {symbol}: {e}")
        return symbol, None

async def batch_backtest(symbols: list, years: int = 5, max_workers: int = 8):
    """Run parallel batch backtesting"""
    logger.info(f"Starting batch backtest: {len(symbols)} symbols, {years} years, {max_w

    # Create semaphore to limit concurrent backtests
    semaphore = asyncio.Semaphore(max_workers)

    async def backtest_with_limit(symbol):
        async with semaphore:
            return await backtest_symbol(symbol, years)

    # Run all backtests in parallel
    results = await asyncio.gather(*[backtest_with_limit(s) for s in symbols])

    # Process results
    successful = [r for r in results if r[1] is not None]
    failed = [r[0] for r in results if r[1] is None]

    logger.info(f"Completed: {len(successful)} successful, {len(failed)} failed")

    return dict(successful), failed

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("--symbols", nargs="+", required=True)
    parser.add_argument("--years", type=int, default=5)
    parser.add_argument("--workers", type=int, default=8)
    args = parser.parse_args()

    results, failed = asyncio.run(batch_backtest(args.symbols, args.years, args.workers)

    # Save results
    import json
    output_file = f"/home/argo/workspace/backtest_results/batch_{datetime.now().strftime
    with open(output_file, 'w') as f:
        json.dump(results, f, indent=2, default=str)

    print(f"Results saved to: {output_file}")
```

1.6 Remote Access Setup

```
# On MacBook: Create SSH config
cat >> ~/.ssh/config << 'EOF'
Host backtest-vps
    HostName <backtest-vps-ip>
    User argo
    IdentityFile ~/.ssh/id_rsa
    ServerAliveInterval 60
    ServerAliveCountMax 3
EOF

# Test connection
ssh backtest-vps "echo 'Connection successful'"
```

1.7 Helper Scripts (MacBook)

```
# Create helper script on MacBook
cat > scripts/backtest_remote.sh << 'EOF'
#!/bin/bash
# Run backtest on VPS from MacBook
```

```
SYMBOLS="${1:-AAPL,NVDA,TSLA}"
YEARS="${2:-5}"
WORKERS="${3:-8}"

echo "  Running backtest on VPS..."
echo "   Symbols: $SYMBOLS"
echo "   Years: $YEARS"
echo "   Workers: $WORKERS"

ssh backtest-vps "cd /home/argo/workspace/argo-alpine-workspace && \
    source argo/venv/bin/activate && \
    python scripts/backtest_vps/batch_backtest_parallel.py \
    --symbols $(echo $SYMBOLS | tr ',' ' ') \
    --years $YEARS \
    --workers $WORKERS"

echo "  Backtest complete! Results on VPS: /home/argo/workspace/backtest_results/"
EOF

chmod +x scripts/backtest_remote.sh
```

Phase 2: Dev VPS Setup

2.1 VPS Specifications

Recommended Provider: Hetzner

Specifications: - **CPU**: 4 cores - **RAM**: 8GB - **Storage**: 100GB NVMe SSD - **Network**: 1Gbps - **Cost**: ~$15/month

2.2 Initial Setup

```
# Similar to backtesting VPS but lighter
ssh root@dev-vps-ip
```

```
# Install base tools
apt update && apt upgrade -y
apt install -y git python3.11 python3.11-venv python3-pip \
    docker.io docker-compose build-essential \
    postgresql-client redis-tools nginx

# Setup Docker
systemctl enable docker
systemctl start docker
usermod -aG docker argo
```

2.3 Full Dev Stack Setup

```
# Clone repository
su - argo
cd /home/argo/workspace
git clone <your-repo-url> argo-alpine-workspace
cd argo-alpine-workspace

# Setup Argo dev environment
cd argo
python3.11 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Setup Alpine backend
cd ../alpine-backend
python3.11 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Setup Alpine frontend
cd ../alpine-frontend
```

```
npm install

# Start Docker services (PostgreSQL, Redis)
cd ..
docker-compose -f alpine-backend/docker-compose.local.yml up -d
```

2.4 Systemd Services for Dev Stack

```
# Create systemd service for Argo dev
sudo tee /etc/systemd/system/argo-dev.service << 'EOF'
[Unit]
Description=Argo Development API
After=network.target

[Service]
Type=simple
User=argo
WorkingDirectory=/home/argo/workspace/argo-alpine-workspace/argo
Environment="PATH=/home/argo/workspace/argo-alpine-workspace/argo/venv/bin"
ExecStart=/home/argo/workspace/argo-alpine-workspace/argo/venv/bin/uvicorn main:app --ho
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF

# Create systemd service for Alpine backend dev
sudo tee /etc/systemd/system/alpine-backend-dev.service << 'EOF'
[Unit]
Description=Alpine Backend Development API
After=network.target postgresql.service

[Service]
```

```
Type=simple
User=argo
WorkingDirectory=/home/argo/workspace/argo-alpine-workspace/alpine-backend
Environment="PATH=/home/argo/workspace/argo-alpine-workspace/alpine-backend/venv/bin"
ExecStart=/home/argo/workspace/argo-alpine-workspace/alpine-backend/venv/bin/uvicorn bac
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF

# Enable and start services
sudo systemctl daemon-reload
sudo systemctl enable argo-dev alpine-backend-dev
sudo systemctl start argo-dev alpine-backend-dev
```

2.5 CI/CD Setup (Optional)

```
# Setup GitHub Actions runner (if using GitHub)
mkdir -p /home/argo/actions-runner
cd /home/argo/actions-runner
curl -o actions-runner-linux-x64-2.311.0.tar.gz -L https://github.com/actions/runner/rel
tar xzf ./actions-runner-linux-x64-2.311.0.tar.gz
./config.sh --url <repo-url> --token <token>
sudo ./svc.sh install
sudo ./svc.sh start
```

Phase 3: Integration & Workflow

3.1 Optimized Development Workflow

```
                   OPTIMIZED WORKFLOW
```

```
1. Code on MacBook
     Edit code (best IDE experience)
     Quick local test
     Git commit/push

2. Dev VPS (24/7)
     Auto-pull on git push (webhook)
     Run full integration tests
     Long-running tests
     CI/CD pipeline

3. Backtesting VPS
     Heavy backtesting (on-demand)
     Parallel processing
     Overnight batches

4. Production Deployment
     Deploy from MacBook
     Blue-green deployment
     Health checks
```

3.2 Automated Sync Script

```bash
# On Dev VPS: Auto-sync on git push
cat > /home/argo/workspace/argo-alpine-workspace/scripts/sync_from_git.sh << 'EOF'
#!/bin/bash
# Auto-sync script for Dev VPS

cd /home/argo/workspace/argo-alpine-workspace
```

```
# Pull latest changes
git pull origin main

# Update Argo
cd argo
source venv/bin/activate
pip install -r requirements.txt --quiet
deactivate

# Update Alpine backend
cd ../alpine-backend
source venv/bin/activate
pip install -r requirements.txt --quiet
deactivate

# Update Alpine frontend
cd ../alpine-frontend
npm install --quiet

# Restart services
sudo systemctl restart argo-dev alpine-backend-dev

echo "  Dev VPS synced and restarted"
EOF

chmod +x /home/argo/workspace/argo-alpine-workspace/scripts/sync_from_git.sh
```

3.3 Backtesting Workflow Integration

```
# On MacBook: Enhanced backtesting workflow
cat > scripts/backtest_workflow.sh << 'EOF'
#!/bin/bash
# Complete backtesting workflow
```

```
SYMBOL="${1:-AAPL}"
YEARS="${2:-5}"

echo "  Quick local test (1 year)..."
cd argo
source venv/bin/activate
python -c "
import asyncio
from argo.backtest.strategy_backtester import StrategyBacktester
from datetime import datetime, timedelta

async def quick_test():
    bt = StrategyBacktester()
    end = datetime.now()
    start = end - timedelta(days=365)
    result = await bt.run_backtest('$SYMBOL', start_date=start, end_date=end)
    print(f'Quick test: Win rate = {result.win_rate_pct:.2f}%')

asyncio.run(quick_test())
"
deactivate

echo ""
echo "  Running full backtest on VPS ($YEARS years)..."
./scripts/backtest_remote.sh "$SYMBOL" "$YEARS" 8

echo ""
echo "  Downloading results..."
scp backtest-vps:/home/argo/workspace/backtest_results/batch_*.json ./backtest_results/
EOF
```

```
chmod +x scripts/backtest_workflow.sh
```

Phase 4: Monitoring & Management

4.1 Monitoring Setup

```
# On both VPS: Install monitoring
# Use existing Prometheus/Grafana or simple monitoring

cat > /home/argo/workspace/monitor_vps.sh << 'EOF'
#!/bin/bash
# Simple VPS monitoring script

echo "=== VPS Status ==="
echo "CPU: $(top -bn1 | grep "Cpu(s)" | awk '{print $2}')"
echo "Memory: $(free -h | grep Mem | awk '{print $3 "/" $2}')"
echo "Disk: $(df -h / | tail -1 | awk '{print $5}')"
echo ""
echo "=== Services ==="
systemctl is-active argo-dev && echo "  Argo Dev: Active" || echo "  Argo Dev: Inactive"
systemctl is-active alpine-backend-dev && echo "  Alpine Dev: Active" || echo "  Alpine I
EOF

chmod +x /home/argo/workspace/monitor_vps.sh
```

4.2 Backup Strategy

```
# Automated backups for VPS
cat > /home/argo/workspace/backup_vps.sh << 'EOF'
#!/bin/bash
# Backup VPS data

BACKUP_DIR="/home/argo/backups"
DATE=$(date +%Y%m%d_%H%M%S)

mkdir -p "$BACKUP_DIR"

# Backup backtest results
```

```
if [ -d "/home/argo/workspace/backtest_results" ]; then
    tar -czf "$BACKUP_DIR/backtest_results_$DATE.tar.gz" \
        /home/argo/workspace/backtest_results
fi

# Backup databases
if [ -f "/home/argo/workspace/backtest_data/signals.db" ]; then
    sqlite3 /home/argo/workspace/backtest_data/signals.db \
        ".backup $BACKUP_DIR/signals_$DATE.db.backup"
fi

# Keep only last 7 days
find "$BACKUP_DIR" -name "*.tar.gz" -mtime +7 -delete
find "$BACKUP_DIR" -name "*.db.backup" -mtime +7 -delete

echo "  Backup complete: $BACKUP_DIR"
EOF

# Add to cron (daily at 2 AM)
(crontab -l 2>/dev/null; echo "0 2 * * * /home/argo/workspace/backup_vps.sh") | crontab
```

Before & After Analysis

Performance Comparison

Task	Before (MacBook)	After (VPS)	Improvement
Single symbol, 5 years	5-10 min	3-5 min	**2x faster**
10 symbols, 5 years (sequential)	50-100 min	15-20 min	**3-5x faster**

Task	Before (MacBook)	After (VPS)	Improvement
10 symbols, 5 years (parallel)	30-60 min	10-15 min	**3-4x faster**
50 symbols, 5 years (parallel)	N/A (too slow)	30-45 min	**∞ (enables)**
Overnight batch (100 symbols)	N/A	2-3 hours	**∞ (enables)**
Strategy optimization	8+ hours	2-3 hours	**3-4x faster**
Integration tests	Manual, slow	Automated, fast	**10x faster**

Development Workflow Comparison

Aspect	Before	After
Code editing	MacBook (good)	MacBook (same)
Quick testing	MacBook (fast)	MacBook (same)
Heavy backtesting	MacBook (slow)	VPS (fast)
Overnight processing	Impossible	VPS (enabled)
Integration testing	Manual	Dev VPS (automated)
CI/CD	None	Dev VPS (enabled)
24/7 dev environment	No	Dev VPS (yes)

Cost Analysis

Component	Monthly Cost	Benefit
Backtesting VPS	$30-35	3-5x faster backtesting, enables parallel
Dev VPS	$15-20	24/7 dev, CI/CD, integration testing
Total	**$45-55**	**Significant productivity gains**

ROI Calculation: - Time saved: 20-30 hours/month on backtesting - Cost: $45-55/month - Value: $1.50-2.75/hour (if time = $50/hour, saves $1,000-1,500/month)

Optimal Use Cases

Backtesting VPS Use Cases

1. **Parallel Batch Backtesting**

```
# Test 50 symbols simultaneously
./scripts/backtest_remote.sh "AAPL,NVDA,TSLA,..." 5 8
```

2. **Overnight Strategy Optimization**

```
# Run grid search overnight
ssh backtest-vps "python scripts/optimize_strategy.py --parallel 8"
```

3. **Walk-Forward Analysis**

```
# Rolling window backtests
ssh backtest-vps "python scripts/walk_forward.py --windows 20"
```

4. **Multi-Year Analysis**

```
# Test strategies over 10 years
./scripts/backtest_remote.sh "AAPL" 10 1
```

5. **Parameter Sensitivity Analysis**

```
# Test different parameters
ssh backtest-vps "python scripts/parameter_sweep.py"
```

Dev VPS Use Cases

1. **24/7 Integration Testing**

```
# Automated tests on every push
git push → Dev VPS auto-tests
```

2. **Long-Running Tests**

```
# Tests that take hours
ssh dev-vps "pytest tests/integration/long_running/ -v"
```

3. **CI/CD Pipeline**

```
# Automated deployment testing
git push → Dev VPS → Test → Deploy to staging
```

4. **Shared Dev Environment**

```
# Team can access same dev environment
ssh dev-vps "cd workspace && git pull"
```

5. **Performance Testing**

```
# Load testing without affecting MacBook
ssh dev-vps "python scripts/load_test.py"
```

Implementation Timeline

Week 1: Backtesting VPS

- **Day 1-2**: Provision and setup
- **Day 3-4**: Configure environment
- **Day 5**: Test and optimize
- **Day 6-7**: Integrate into workflow

Month 2: Dev VPS

- **Week 1**: Provision and setup
- **Week 2**: Configure dev stack
- **Week 3**: Setup CI/CD
- **Week 4**: Integrate and test

Success Metrics

Backtesting VPS

- □ 3-5x faster backtesting achieved
- □ Can run 50+ symbols in parallel
- □ Overnight batches working
- □ Strategy optimization < 3 hours

Dev VPS

- □ 24/7 dev environment running
- □ Automated tests on git push
- □ CI/CD pipeline functional
- □ Integration tests automated

Recommendations

Final Recommendation: YES - Implement Both VPS Servers

Priority 1: Backtesting VPS (Immediate) - **Why**: Biggest performance gain (3-5x faster) - **Impact**: Enables parallel processing, overnight batches - **Risk**: Low (isolated from production) - **ROI**: Immediate

Priority 2: Dev VPS (Within 1-2 Months) - **Why**: Enables 24/7 dev, CI/CD - **Impact**: Better testing, automation - **Risk**: Low - **ROI**: Medium-term

Implementation Strategy

1. **Start with Backtesting VPS** (Week 1)
   - Immediate performance gains
   - Low risk
   - High ROI
2. **Add Dev VPS** (Month 2)
   - After backtesting VPS proven
   - When CI/CD needed
   - When team grows

Expected Outcomes

- **3-5x faster backtesting**
- **Parallel processing enabled**
- **Overnight processing enabled**
- **24/7 dev environment**
- **CI/CD automation**
- **20-30 hours/month saved**

Conclusion

The implementation of Backtesting VPS and Dev VPS infrastructure will significantly improve development and testing workflows, enabling faster iteration, better testing coverage, and more efficient resource utilization. The investment of $45-55/month provides substantial ROI through time savings and improved capabilities.

Total Investment: $45-55/month
Expected ROI: 20-30 hours/month saved, 3-5x faster backtesting, enables new capabilities

This setup provides: - Faster backtesting (3-5x) - Parallel processing (enables new capabilities) - Overnight processing (enables new workflows) - 24/7 dev environment (improves testing) - CI/CD automation (improves quality)

Document Version: 1.0
Last Updated: January 2025
Status: Implementation Plan