HashInsight

Operations Manual for HashInsight Platform

: v2.0

: 2025-10-03

: HashInsight Platform Operations Team

: INTERNAL - CONFIDENTIAL

1

1.1

HashInsight

Web	Flask	3.0+	Python Web
WSGI	Gunicorn	21.0+	HTTP
	PostgreSQL	15+	(Neon)
	Redis /	7.0+	
	Web3.py + Base L2	-	
	Cryptography	41.0+	

1.2

```
HashInsight Platform
├── Core Application (main.py + app.py)
│   ├── Authentication & Authorization
│   ├── Session Management
│   └── Security Middleware
│
├── Mining Management Module
│   ├── Miner Dashboard
│   ├── Batch Calculator
│   └── Analytics Engine
│
├── CRM & Client Module
│   ├── Customer Management
│   ├── Billing System
│   └── Subscription Management
│
├── Blockchain Integration Module
│   ├── SLA NFT Management
│   ├── Verifiable Computing
│   └── Trust Reconciliation
│
└── Admin & Analytics Module
    ├── Market Data Analysis
    ├── Performance Monitoring
    └── Reporting System
```

1.3

- **KMS** : AWS KMS GCP KMS Azure Key Vault
- **mTLS** : CRL/OCSP
- **API** : `hsi_dev_key_*`
- **WireGuard VPN**:
- : SOC 2 / PCI DSS / GDPR

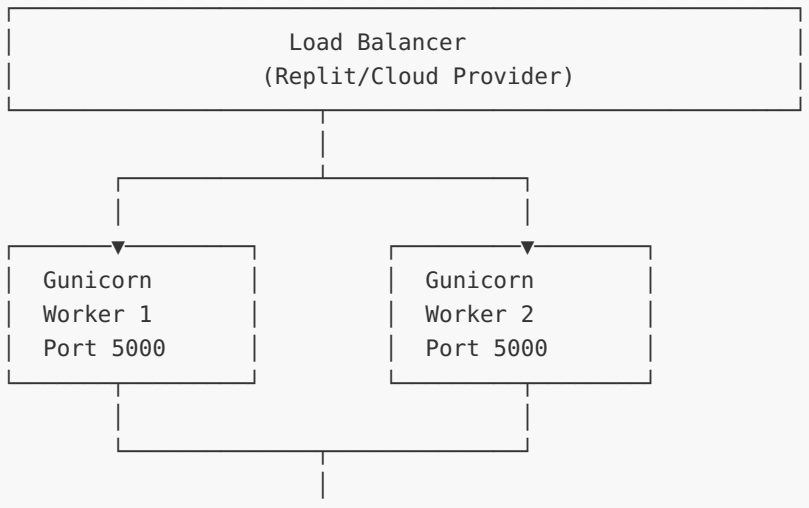
SLO

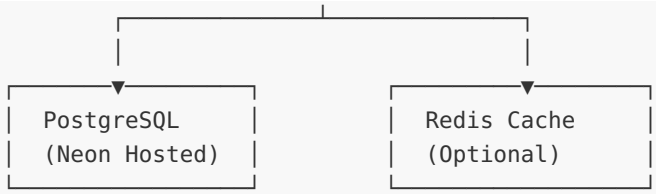
- : ≥99.95% (≤21.6 /)
- : p95 ≤250ms
- : ≤0.1%
- **Prometheus** :
- :

⚡

- **Request Coalescing**: 9.8
- : Redis +
- : PostgreSQL
- : 5000+

1.4





2

2.1

`DATABASE_URL`	String	PostgreSQL	`postgresql://user:pass@host:5432/db`
`SESSION_SECRET`	String	Flask (≥32)	`your-secure-random-secret-key-here`
`ENCRYPTION_PASSWORD`	String	(≥32)	`encryption-master-key-32-chars-min`

```
# .env
DATABASE_URL=postgresql://hashinsight_user:secure_password@neon-host.us-east-1.aws.neon.tech:5432/hash
SESSION_SECRET=generate_with_python_secrets_token_urlsafe_32
ENCRYPTION_PASSWORD=generate_with_python_secrets_token_urlsafe_32
```

```
# Python
import secrets
print(f"SESSION_SECRET={secrets.token_urlsafe(32)}")
print(f"ENCRYPTION_PASSWORD={secrets.token_urlsafe(32)}")
```

2.2

`BLOCKCHAIN_ENABLED`	Boolean	`false`	
`BLOCKCHAIN_PRIVATE_KEY`	String	-	(0x)
`BLOCKCHAIN_NETWORK`	String	`base-sepolia`	
`BASE_RPC_URL`	String	`https://sepolia.base.org`	Base L2 RPC

```
#
BLOCKCHAIN_ENABLED=true
BLOCKCHAIN_PRIVATE_KEY=0x1234567890abcdef...
BLOCKCHAIN_NETWORK=base-sepolia
```

2.3

`BACKUP_DIR`	String	`/tmp/backups`	
`BACKUP_ENCRYPTION_KEY`	String	-	
`BACKUP_RETENTION_DAYS`	Integer	`30`	
`BACKUP_STORAGE_TYPE`	String	`local`	(s3/azure/gcs)
`BACKUP_STORAGE_BUCKET`	String	-	

```
# AWS S3
BACKUP_DIR=/var/backups/hashinsight
BACKUP_ENCRYPTION_KEY=backup-encryption-key-32-chars
BACKUP_RETENTION_DAYS=30
BACKUP_STORAGE_TYPE=s3
BACKUP_STORAGE_BUCKET=hashinsight-backups
BACKUP_STORAGE_REGION=us-east-1
AWS_ACCESS_KEY_ID=AKIAXXXXXXXX
AWS_SECRET_ACCESS_KEY=xxxxxxxxxx
```

2.4 KMS

AWS KMS

```
AWS_KMS_KEY_ID=arn:aws:kms:us-east-1:123456789:key/xxxxx
AWS_KMS_REGION=us-east-1
AWS_ACCESS_KEY_ID=AKIAXXXXXXXX
AWS_SECRET_ACCESS_KEY=xxxxxxxxxx
```

GCP KMS

```
GCP_KMS_PROJECT_ID=hashinsight-prod
GCP_KMS_LOCATION=us-east1
GCP_KMS_KEYRING=hashinsight-keyring
GCP_KMS_KEY_ID=encryption-key
GOOGLE_APPLICATION_CREDENTIALS=/path/to/service-account.json
```

Azure Key Vault

```
AZURE_KEY_VAULT_URL=https://hashinsight-vault.vault.azure.net/
AZURE_TENANT_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
AZURE_CLIENT_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
AZURE_CLIENT_SECRET=xxxxxxxxxx
```

2.5

`ENABLE_BACKGROUND_SERVICES`	Boolean	`false`	'1'
`PROMETHEUS_PORT`	Integer	`9090`	Prometheus
`SLO_MEASUREMENT_WINDOW`	Integer	`30`	SLO ( )

: `FAST_STARTUP` `main.py` 3.4

2.6 API

`COINWARZ_API_KEY`	CoinWarz API	https://www.coinwarz.com/api
`COINGECKO_API_KEY`	CoinGecko API	https://www.coingecko.com/api
`SENDGRID_API_KEY`	SendGrid	https://sendgrid.com

2.7

```
HashInsight/
├── config.py                    #  ()
├── .env                         #  (Git)
├── .env.example                 #
└── replit.md                   #
```

2.8

```
#
python -c "
import os
required = ['DATABASE_URL', 'SESSION_SECRET', 'ENCRYPTION_PASSWORD']
missing = [v for v in required if not os.getenv(v)]
if missing:
    print(f' Missing: {missing}')
    exit(1)
print(' All required variables set')
"
```

3

3.1

( )

```
gunicorn --bind 0.0.0.0:5000 --reuse-port --reload main:app
```

`--bind 0.0.0.0:5000`	5000 (Replit)
`--reuse-port`	worker
`--reload`	( )
`--workers 4`	Worker (CPU ×2+1)
`--timeout 120`	Worker ( )

```
gunicorn \
  --bind 0.0.0.0:5000 \
  --workers 4 \
  --threads 2 \
  --worker-class gthread \
  --timeout 120 \
  --max-requests 1000 \
  --max-requests-jitter 50 \
  --access-logfile - \
  --error-logfile - \
  --log-level info \
  --preload \
  main:app
```

3.2

⚠ : 5000

```
#
lsof -i :5000

#
kill -9 $(lsof -t -i:5000)
```

3.3

```
#
curl http://localhost:5000/health

#  ()
curl http://localhost:5000/health/detailed

#
{
  "status": "healthy",
  "timestamp": "2025-10-03T12:00:00Z",
  "database": "connected",
  "cache": "available",
  "version": "2.0.0"
}
```

3.4 Fast Startup

HashInsight `main.py` `config.py`

```
# main.py
fast_startup = os.environ.get("FAST_STARTUP", "1").lower() in ("1", "true", "yes")  #
skip_db_check = os.environ.get("SKIP_DATABASE_HEALTH_CHECK", "1").lower() in ("1", "true", "yes")  #
```

`config.py`

```
#  ()
export FAST_STARTUP=1
export SKIP_DATABASE_HEALTH_CHECK=1

#  ()
export FAST_STARTUP=0
export SKIP_DATABASE_HEALTH_CHECK=0
```

⚠ - `config.py` - `main.py` - `config.py`

Fast Startup

1. (2-3) 2. 5 `ENABLE_BACKGROUND_SERVICES=1` 3. 4.

CI/CD

1. 2. 3. 10-15 4.

3.5

⚠ : ORM **SQL**

```
#
#  app.py  db.create_all()

#
python -c "from app import app, db; app.app_context().push(); db.create_all()"
```

3.6

```
# Step 1: Canary
# 1worker
gunicorn --bind 0.0.0.0:5001 --workers 1 main:app

# Step 2: Canary (5-10)
watch -n 5 'curl -s http://localhost:5001/health | jq'

# Step 3:
# : old(90%) -> new(10%)
#

# Step 4:
# old(0%) -> new(100%)
```

```
killall -9 gunicorn  #
gunicorn --bind 0.0.0.0:5000 --workers 4 main:app  #
```

3.7

```
# SIGTERM ()
kill -TERM $(cat /var/run/gunicorn.pid)

# 30
sleep 30

#  ()
kill -KILL $(cat /var/run/gunicorn.pid)
```

3.8

```
#
export LOG_LEVEL=INFO
export LOG_FORMAT=json

#
tail -f /var/log/hashinsight/app.log

#
grep ERROR /var/log/hashinsight/app.log | tail -20

# Replit
# Replit Console
```

3.9

- [ ] (DATABASE_URL, SESSION_SECRET, ENCRYPTION_PASSWORD)
- [ ]
- [ ] 5000
- [ ] 200
- [ ] (`logs/audit.jsonl`)
- [ ]
- [ ] SSL (mTLS)
- [ ] KMS ( )
- [ ] Prometheus (`:9090/metrics`)

4

4.1 SLO

HashInsight SLO

SLO

	≥99.95%	≤21.6 /	30
	≥99.9%	≤43.2 /	30

SLO

P50	≤100ms	5
P95	≤250ms	5
P99	≤500ms	5

SLO

4xx	≤1%	
5xx	≤0.1%	

4.2 Prometheus

```
# monitoring/prometheus_exporter.py
from prometheus_client import Counter, Histogram, Gauge

#
request_count = Counter(
    'hashinsight_requests_total',
    'Total request count',
    ['method', 'endpoint', 'status']
)

#
request_latency = Histogram(
    'hashinsight_request_latency_seconds',
    'Request latency',
    ['method', 'endpoint'],
    buckets=[0.01, 0.05, 0.1, 0.25, 0.5, 1.0, 2.5, 5.0]
)

#
cache_hit_rate = Gauge(
    'hashinsight_cache_hit_rate',
    'Cache hit rate percentage'
)

#
db_query_duration = Histogram(
    'hashinsight_db_query_duration_seconds',
    'Database query duration',
    ['query_type'],
    buckets=[0.001, 0.01, 0.05, 0.1, 0.5, 1.0]
)

# SLO
slo_compliance = Gauge(
    'hashinsight_slo_compliance',
    'SLO compliance percentage',
    ['slo_type']
)
```

```
# Prometheus
curl http://localhost:9090/metrics

#
# HELP hashinsight_requests_total Total request count
```

```
# TYPE hashinsight_requests_total counter
hashinsight_requests_total{method="GET",endpoint="/dashboard",status="200"} 1234

# HELP hashinsight_request_latency_seconds Request latency
# TYPE hashinsight_request_latency_seconds histogram
hashinsight_request_latency_seconds_bucket{method="GET",endpoint="/api/miners",le="0.1"} 450
hashinsight_request_latency_seconds_bucket{method="GET",endpoint="/api/miners",le="0.25"} 480
```

4.3 Grafana

```
{
  "dashboard": {
    "title": "HashInsight Production Monitoring",
    "panels": [
      {
        "title": "Request Rate",
        "targets": [{
          "expr": "rate(hashinsight_requests_total[5m])"
        }]
      },
      {
        "title": "P95 Latency",
        "targets": [{
          "expr": "histogram_quantile(0.95, rate(hashinsight_request_latency_seconds_bucket[5m]))"
        }]
      },
      {
        "title": "Error Rate",
        "targets": [{
          "expr": "rate(hashinsight_requests_total{status=~\"5..\"}[5m]) / rate(hashinsight_requests_
        }]
      },
      {
        "title": "SLO Compliance",
        "targets": [{
          "expr": "hashinsight_slo_compliance"
        }]
      }
    ]
  }
}
```

4.4

Prometheus

```
# prometheus/alerts.yml
groups:
  - name: hashinsight_alerts
    interval: 30s
    rules:
      #
      - alert: HighErrorRate
        expr: |
          rate(hashinsight_requests_total{status=~"5.."}[5m])
          / rate(hashinsight_requests_total[5m]) > 0.01
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "High 5xx error rate detected"
          description: "Error rate is {{ $value | humanizePercentage }}"

      #
      - alert: HighLatency
        expr: |
          histogram_quantile(0.95,
            rate(hashinsight_request_latency_seconds_bucket[5m])
          ) > 0.25
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "P95 latency exceeds SLO"
          description: "P95 latency is {{ $value }}s (SLO: 0.25s)"

      # SLO
      - alert: ErrorBudgetExhausted
        expr: hashinsight_slo_error_budget_remaining < 0.1
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "SLO error budget nearly exhausted"
          description: "Only {{ $value | humanizePercentage }} budget remaining"

      #
      - alert: DatabaseConnectionFailure
        expr: hashinsight_db_connection_status == 0
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "Database connection failure"
```

```
          description: "Unable to connect to PostgreSQL"

      #
      - alert: LowCacheHitRate
        expr: hashinsight_cache_hit_rate < 50
        for: 15m
        labels:
          severity: warning
        annotations:
          summary: "Low cache hit rate"
          description: "Cache hit rate is {{ $value }}%"
```

4.5

HashInsight

```
# monitoring/circuit_breaker.py
from monitoring.circuit_breaker import CircuitBreaker, circuit_breaker

#
db_breaker = CircuitBreaker(
    failure_threshold=5,      # 5
    recovery_timeout=60,      # 60
    name="database_queries"
)

# API
@circuit_breaker(
    failure_threshold=3,
    recovery_timeout=30,
    name="external_api"
)
def call_external_api():
    response = requests.get("https://api.coinwarz.com/...")
    return response.json()
```

```
#
curl http://localhost:5000/api/circuit-breakers

#
{
  "database_queries": {
    "state": "closed",
    "failure_count": 0,
    "total_calls": 1234,
    "success_rate": "99.8%"
  },
```

```
  "external_api": {
    "state": "half_open",
    "failure_count": 3,
    "total_calls": 456,
    "success_rate": "95.2%"
  }
}
```

4.6

Slack

```
#
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00/B00/xxxxx
ALERT_SLACK_CHANNEL=#hashinsight-alerts
```

PagerDuty

```
PAGERDUTY_API_KEY=xxxxx
PAGERDUTY_SERVICE_KEY=xxxxx
```

4.7

- [ ] Prometheus （`:9090/targets`）
- [ ] Grafana
- [ ]
- [ ] Slack/PagerDuty
- [ ] SLO
- [ ]

5

5.1

HashInsight `backup/backup_manager.py`

- **PostgreSQL** (pg_dump)
- **AES-256** ( )
- **gzip** ( )
- (S3/Azure/GCS)
- (SHA256)

```
# cron
# /etc/cron.d/hashinsight-backup

# 2
0 2 * * * /usr/bin/python3 /app/backup/backup_manager.py --type full

# 4
0 */4 * * * /usr/bin/python3 /app/backup/backup_manager.py --type incremental

# 3
0 3 * * 0 /usr/bin/python3 /app/backup/backup_manager.py --upload
```

5.2

```
#
python backup/backup_manager.py

#
 : hashinsight_backup_20251003_140000.sql.gz.enc
 : 245.3 MB
 : AES-256
 : e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
  : 45.2s
```

5.3

	30	
	7	4
	12	
	12	1

```
# 30
python backup/backup_manager.py --cleanup --days 30

#
  : hashinsight_backup_20250903_*.sql.gz.enc
 :  2.1 GB
```

5.4 RTO/RPO

RTO ( )	≤4	~2
RPO ( )	≤15	~4

5.5

Step 1:

```
#
ls -lh /tmp/backups/

#  (S3)
aws s3 ls s3://hashinsight-backups/
```

Step 2: ( )

```
# S3
aws s3 cp s3://hashinsight-backups/hashinsight_backup_20251003_020000.sql.gz.enc /tmp/restore/

# Azure
az storage blob download \
  --account-name hashinsight \
  --container-name backups \
  --name hashinsight_backup_20251003_020000.sql.gz.enc \
  --file /tmp/restore/backup.sql.gz.enc
```

Step 3:

```
# backup_manager
python backup/backup_manager.py \
  --decrypt /tmp/restore/hashinsight_backup_20251003_020000.sql.gz.enc \
  --output /tmp/restore/backup.sql.gz

#  ()
openssl enc -d -aes-256-cbc \
  -in hashinsight_backup_20251003_020000.sql.gz.enc \
  -out backup.sql.gz \
  -pass env:BACKUP_ENCRYPTION_KEY
```

Step 4:

```
gunzip /tmp/restore/backup.sql.gz
# : backup.sql
```

Step 5:

```
# ⚠ : !
#

# PostgreSQL
psql $DATABASE_URL < /tmp/restore/backup.sql

#
psql $DATABASE_URL -c "SELECT COUNT(*) FROM users;"
psql $DATABASE_URL -c "SELECT COUNT(*) FROM miners;"
```

Step 6:

```
#
systemctl restart hashinsight

#
curl http://localhost:5000/health

#
curl http://localhost:5000/api/miners | jq '.count'
```

5.6

:

1.

```
#
createdb hashinsight_dr_test
export DATABASE_URL=postgresql://localhost/hashinsight_dr_test
```

1.

```
#
psql $DATABASE_URL -c "DROP SCHEMA public CASCADE; CREATE SCHEMA public;"
```

1. (5.5)
2.

```
#
python -c "
from app import app, db
from models import User, Miner
with app.app_context():
    assert db.session.query(User).count() > 0
    assert db.session.query(Miner).count() > 0
    print(' ')
"
```

1. - - - -

5.7

```
#
stat -c '%y' /tmp/backups/hashinsight_backup_*.sql.gz.enc | tail -1

#
du -h /tmp/backups/hashinsight_backup_*.sql.gz.enc

#
python backup/backup_manager.py --verify /tmp/backups/hashinsight_backup_20251003_020000.sql.gz.enc
```

6

6.1 KMS

HashInsight KMS (`common/crypto/envelope.py`)

```
┌──────────────────────────────────────────────┐
│  1.                          │
│  2. KMS(DEK)                 │
│  3. DEK                          │
│  4. KMS(CMK)DEK                  │
│  5. :  + DEK                 │
└──────────────────────────────────────────────┘

   ▲
   │ KMS
   ▼
┌──────────────────────────────────────────────┐
│  :                                   │
│  1.  + DEK                   │
│  2. KMSDEK                           │
│  3. DEK                      │
└──────────────────────────────────────────────┘
```

KMS

AWS KMS

```
# AWS KMS
from common.crypto.envelope import KMSClient, KMSProvider, EncryptionContext
```

```
kms_config = {
    'key_id': 'arn:aws:kms:us-east-1:123456789:key/xxxxx',
    'region': 'us-east-1'
}

client = KMSClient(KMSProvider.AWS_KMS, kms_config)

#
context = EncryptionContext(
    purpose="user_data_encryption",
    tenant_id="tenant_123"
)

ciphertext = client.encrypt_secret(
    plaintext="sensitive data",
    key_id=kms_config['key_id'],
    context=context
)
```

GCP KMS

```
# GCP KMS
kms_config = {
    'project_id': 'hashinsight-prod',
    'location': 'us-east1',
    'keyring': 'hashinsight-keyring',
    'key_id': 'encryption-key'
}

client = KMSClient(KMSProvider.GCP_KMS, kms_config)
```

Azure Key Vault

```
# Azure Key Vault
kms_config = {
    'vault_url': 'https://hashinsight-vault.vault.azure.net/',
    'key_name': 'encryption-key'
}

client = KMSClient(KMSProvider.AZURE_KEY_VAULT, kms_config)
```

```
# 1. KMS
aws kms create-key --description "HashInsight Master Key v2"

# 2.
```

```
export AWS_KMS_KEY_ID=arn:aws:kms:us-east-1:123456789:key/new-key-id

# 3.  ()
python scripts/rotate_encryption_keys.py --old-key OLD_KEY_ID --new-key NEW_KEY_ID

# 4.
python scripts/verify_encryption.py --key-id NEW_KEY_ID

# 5.  (90)
aws kms disable-key --key-id OLD_KEY_ID
```

6.2 mTLS

```
# 1. CA
openssl genrsa -out ca.key 4096
openssl req -new -x509 -days 3650 -key ca.key -out ca.crt \
  -subj "/C=US/O=HashInsight/CN=HashInsight Root CA"

# 2.
openssl genrsa -out server.key 4096
openssl req -new -key server.key -out server.csr \
  -subj "/C=US/O=HashInsight/CN=*.hashinsight.net"
openssl x509 -req -days 365 -in server.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out server.crt

# 3.
openssl genrsa -out client.key 4096
openssl req -new -key client.key -out client.csr \
  -subj "/C=US/O=HashInsight/CN=client.hashinsight.net"
openssl x509 -req -days 365 -in client.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out client.crt
```

mTLS

```
#
export MTLS_ENABLED=true
export MTLS_CA_CERT_PATH=/app/certs/ca.crt
export MTLS_SERVER_CERT_PATH=/app/certs/server.crt
export MTLS_SERVER_KEY_PATH=/app/certs/server.key
export MTLS_VERIFY_CLIENT=true
export MTLS_ALLOWED_DN_PATTERNS="CN=*.hashinsight.net,O=HashInsight,C=US"
```

mTLS

```
from common.mtls_auth import require_mtls

@app.route('/api/admin/sensitive')
@require_mtls()
def sensitive_endpoint():
    #
    client_dn = g.client_cert_subject
    return jsonify({"message": f"Authenticated as {client_dn}"})
```

6.3 API

HashInsight API : `hsi_{env}_key_{random}`

: - : `hsi_prod_key_a1b2c3d4e5f6g7h8` - : `hsi_dev_key_x9y8z7w6v5u4t3s2`

API

```
#
python scripts/create_api_key.py \
  --user-id 123 \
  --permissions "miners:read,miners:write" \
  --expires-in 90

#
 API
: hsi_prod_key_a1b2c3d4e5f6g7h8
ID: 123
: miners:read,miners:write
: 2025-12-31T23:59:59Z
⚠
```

API

```
# 1.
new_key=$(python scripts/create_api_key.py --user-id 123 --copy-from old_key_id)

# 2.  (7)
#

# 3.
curl -H "Authorization: Bearer $new_key" http://localhost:5000/api/miners
```

```
# 4.
python scripts/revoke_api_key.py --key-id old_key_id
```

6.4 WireGuard

Hub

```
# 1. Hub
sudo bash wireguard/hub_setup.sh

# 2.
sudo ufw allow 51820/udp
sudo ufw enable

# 3. WireGuard
sudo systemctl enable wg-quick@wg0
sudo systemctl start wg-quick@wg0

# 4.
sudo wg show
```

```
# 1.
cd wireguard/site-gateway
python key_manager.py --generate-keys --site beijing-dc1

# 2. Hub
sudo nano /etc/wireguard/wg0.conf

# Peer
[Peer]
PublicKey = site_public_key
AllowedIPs = 10.8.1.0/24
Endpoint = beijing-gateway.hashinsight.net:51820

# 3.
sudo wg-quick down wg0
sudo wg-quick up wg0
```

6.5

HashInsight （`audit/audit_logger.py`）

- (/ /)
- (CRUD)
-
-
-
- API
-

```
{
  "timestamp": "2025-10-03T12:00:00.000Z",
  "event_id": "a1b2c3d4e5f6g7h8",
  "level": "INFO",
  "category": "authentication",
  "action": "login",
  "user_id": "123",
  "user_email": "user@example.com",
  "user_role": "admin",
  "ip_address": "192.168.1.100",
  "status": "success",
  "details": {
    "login_method": "password",
    "two_factor": true
  }
}
```

```
# 100
tail -100 logs/audit.jsonl

#
jq 'select(.user_email == "admin@hashinsight.net")' logs/audit.jsonl

#
jq 'select(.action == "login_failed")' logs/audit.jsonl

# 24
jq --arg date "$(date -d '24 hours ago' -Iseconds)" \
  'select(.timestamp > $date and .level == "SECURITY")' \
  logs/audit.jsonl
```

6.6

HashInsight

SOC 2 Type II

-
- (TLS 1.3)
- (AES-256)
- ( )
-
-

PCI DSS ( )

-
- (KMS)
- (WireGuard)
-

GDPR

-
-
- ( )
-

6.7

- [ ] KMS (/)
- [ ] mTLS ( )
- [ ] API (90)
- [ ] SSL/TLS
- [ ]
- [ ]
- [ ] (SSL)
- [ ] ≥256

- [ ] ( )
- [ ] ( )

7

7.1

```
$ gunicorn main:app
[ERROR] Application failed to start
```

1.

```
# 5000
lsof -i :5000

#
kill -9 $(lsof -t -i:5000)
```

2.

```
#
python3 << 'EOF'
import os
required = ['DATABASE_URL', 'SESSION_SECRET', 'ENCRYPTION_PASSWORD']
for var in required:
    value = os.getenv(var)
    if not value:
        print(f" Missing: {var}")
    else:
        print(f" {var}: {'*' * 8} (set)")
EOF
```

3.

```
# PostgreSQL
psql $DATABASE_URL -c "SELECT version();"
```

```
# Neon
#  https://console.neon.tech
```

4. Python

```
#
python3 -c "
import flask
import gunicorn
import psycopg2
import sqlalchemy
print(' All dependencies OK')
"
```

5.

```
#
export FLASK_DEBUG=1
python3 main.py

#
```

`ModuleNotFoundError: No module named 'flask'`		`pip install -r requirements.txt`
`OperationalError: could not connect to server`		DATABASE_URL Neon
`ValueError: SECRET_KEY must be set`	SESSION_SECRET	`export SESSION_SECRET=$(python -c 'import secrets; print(secrets.token_urlsafe(32))')`
`Address already in use`		`kill -9 $(lsof -t -i:5000)`

7.2

```
sqlalchemy.exc.OperationalError: (psycopg2.OperationalError)
could not connect to server: Connection timed out
```

1.

```
# DATABASE_URL
python3 << 'EOF'
import os
from urllib.parse import urlparse
url = os.getenv('DATABASE_URL')
parsed = urlparse(url)
print(f"Host: {parsed.hostname}")
print(f"Port: {parsed.port}")
print(f"Database: {parsed.path[1:]}")
print(f"User: {parsed.username}")
EOF
```

2.

```
# hostport
export DB_HOST=$(python3 -c "from urllib.parse import urlparse; import os; print(urlparse(os.getenv('
export DB_PORT=$(python3 -c "from urllib.parse import urlparse; import os; print(urlparse(os.getenv('

# TCP
nc -zv $DB_HOST $DB_PORT
```

3.

```
#  (config.py)
SQLALCHEMY_ENGINE_OPTIONS = {
    'pool_size': 10,           # 10
    'pool_recycle': 300,       # 5
    'pool_pre_ping': True,     #
    'pool_timeout': 30,        # 30
    'max_overflow': 20,        # 20
    'connect_args': {
        'connect_timeout': 15  # 15
    }
}
```

4. Neon

```
# Neon
# https://console.neon.tech/app/projects

# :
# - Active () -
# - Idle () -
# - Suspended () -
```

5.

```
# app.py
from sqlalchemy import event, exc
from sqlalchemy.pool import Pool

@event.listens_for(Pool, "connect")
def receive_connect(dbapi_conn, connection_record):
    connection_record.info['pid'] = os.getpid()

@event.listens_for(Pool, "checkout")
def receive_checkout(dbapi_conn, connection_record, connection_proxy):
    pid = os.getpid()
    if connection_record.info['pid'] != pid:
        connection_record.connection = connection_proxy.connection = None
        raise exc.DisconnectionError(
            "Connection record belongs to pid %s, "
            "attempting to check out in pid %s" %
            (connection_record.info['pid'], pid)
        )
```

7.3

-
- 50%
- Redis

1. Redis

```
# Redis
redis-cli ping
# : PONG
```

```
# Redis
redis-cli info memory

#
redis-cli dbsize
```

2.

```
# API
curl http://localhost:5000/api/cache/stats

#
{
  "cache_type": "redis",
  "hit_rate": 75.5,
  "total_requests": 10000,
  "hits": 7550,
  "misses": 2450
}
```

3.

```
# cache_manager.py
if redis_available:
    cache_backend = RedisCache()
else:
    logger.warning("Redis unavailable, falling back to memory cache")
    cache_backend = MemoryCache()
```

4.

```
#
redis-cli FLUSHDB

#
redis-cli --scan --pattern 'hashinsight:*' | xargs redis-cli DEL
```

7.4

- API p95 > 250ms
-

- CPU/

1. Request Coalescing

```
#
curl http://localhost:5000/api/performance/coalescing-stats

#
{
  "enabled": true,
  "performance_improvement": "9.8x",
  "deduplicated_requests": 5432,
  "total_requests": 53210
}
```

2.

```
--  (PostgreSQL)
ALTER DATABASE hashinsight_db SET log_min_duration_statement = 1000;

--
SELECT query, calls, total_time, mean_time
FROM pg_stat_statements
WHERE mean_time > 1000
ORDER BY mean_time DESC
LIMIT 10;
```

3.

```
#
python database/optimize_indexes.py --analyze

#
python database/optimize_indexes.py --suggest
```

4.

```
# CPU
top -b -n 1 | grep gunicorn

#
ps aux | grep gunicorn | awk '{sum+=$6} END {print sum/1024 " MB"}'
```

```
#
psql $DATABASE_URL -c "SELECT count(*) FROM pg_stat_activity WHERE datname = 'hashinsight_db';"
```

5.

```
#
from werkzeug.middleware.profiler import ProfilerMiddleware

app.wsgi_app = ProfilerMiddleware(
    app.wsgi_app,
    restrictions=[10],
    profile_dir='./profiles'
)
```

7.5

```
ERROR:blockchain_integration: : ENCRYPTION_PASSWORD
ERROR:sla_nft_routes: SLA
```

1. ENCRYPTION_PASSWORD

```
#
echo $ENCRYPTION_PASSWORD

#
export ENCRYPTION_PASSWORD=$(python3 -c 'import secrets; print(secrets.token_urlsafe(32))')
```

2.

```
#
python3 << 'EOF'
import os
config = {
    'BLOCKCHAIN_ENABLED': os.getenv('BLOCKCHAIN_ENABLED'),
    'BLOCKCHAIN_PRIVATE_KEY': '***' if os.getenv('BLOCKCHAIN_PRIVATE_KEY') else None,
    'BLOCKCHAIN_NETWORK': os.getenv('BLOCKCHAIN_NETWORK', 'base-sepolia'),
    'BASE_RPC_URL': os.getenv('BASE_RPC_URL', 'https://sepolia.base.org')
}
for k, v in config.items():
    status = '' if v else ''
```

```
    print(f"{status} {k}: {v}")
EOF
```

3. Web3

```
python3 << 'EOF'
from web3 import Web3
import os

rpc_url = os.getenv('BASE_RPC_URL', 'https://sepolia.base.org')
w3 = Web3(Web3.HTTPProvider(rpc_url))

if w3.is_connected():
    print(f" Web3 connected to {rpc_url}")
    print(f"Block number: {w3.eth.block_number}")
else:
    print(f" Web3 connection failed")
EOF
```

4.

```
# 0x64
python3 << 'EOF'
import os
import re

private_key = os.getenv('BLOCKCHAIN_PRIVATE_KEY', '')
if re.match(r'^0x[0-9a-fA-F]{64}$', private_key):
    print(" Private key format valid")
else:
    print(" Invalid private key format")
    print("Expected: 0x + 64 hex characters")
EOF
```

7.6

	(stdout)	
	`logs/audit.jsonl`	JSON Lines
	(stderr)	
Gunicorn	`/var/log/gunicorn/`	
PostgreSQL	Neon Console	
Workflow	`/tmp/logs/`	

```
#
tail -f /var/log/hashinsight/app.log

#
grep ERROR /var/log/hashinsight/app.log | tail -50

#
tail -f logs/audit.jsonl | jq '.'

# Gunicorn
tail -f /var/log/gunicorn/access.log
```

8

8.1

8.1.1

```
#
python database/optimize_indexes.py --auto
```

```
#
python database/optimize_indexes.py --analyze --suggest

# :
#  : 45
#  ...
#  :
#   - CREATE INDEX idx_miners_user_id ON miners(user_id)
#   - CREATE INDEX idx_market_data_timestamp ON market_analytics(created_at DESC)
```

8.1.2

```
--
SELECT
    pid,
    usename,
    application_name,
    client_addr,
    state,
    query_start
FROM pg_stat_activity
WHERE datname = 'hashinsight_db'
ORDER BY query_start;

--
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE datname = 'hashinsight_db'
  AND state = 'idle'
  AND state_change < NOW() - INTERVAL '30 minutes';
```

8.1.3

```
-- pg_stat_statements
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;

-- Top 10
SELECT
    substring(query, 1, 100) AS short_query,
    calls,
    total_time,
    mean_time,
    max_time
FROM pg_stat_statements
WHERE query NOT LIKE '%pg_stat_statements%'
ORDER BY mean_time DESC
LIMIT 10;
```

```
--
SELECT pg_stat_statements_reset();
```

8.1.4

```
#  (90)
python scripts/cleanup_old_data.py --days 90

# :
#  90market_analytics: 12,543
#  90audit: 45,123
#  : 2.3 GB
```

8.2

8.2.1 Redis

```
# HashInsight
redis-cli --scan --pattern 'hashinsight:*' | xargs redis-cli DEL

#
redis-cli --scan --pattern 'hashinsight:miners:*' | xargs redis-cli DEL

#  (Redis)
redis-cli --scan --pattern 'hashinsight:*' | while read key; do
    redis-cli TTL "$key"
done
```

8.2.2

```
# API
curl http://localhost:5000/api/cache/memory-stats

#
{
  "cache_type": "memory",
  "size_mb": 124.5,
  "entries": 5432,
  "hit_rate": 82.3,
  "evictions": 234
}
```

8.2.3 Request Coalescer

```
#
curl http://localhost:5000/api/performance/coalescing-stats | jq

#
curl -X POST http://localhost:5000/api/performance/clear-coalescing-cache
```

8.3

8.3.1

```
# CSV (5000)
# : name,model,hashrate_th,power_w,efficiency

#
python batch/batch_import_manager.py \
  --file miners_upload_5000.csv \
  --user-id 123 \
  --validate

# :
#
#  : 5000
# ⚙  ...
# [████████████████████████████████] 100%
#  : 5000 (: 45.2s)
```

8.3.2

HashInsight 15

```
#
python modules/analytics/engines/analytics_engine.py --collect-now

#
curl http://localhost:5000/api/analytics/collection-status

#
{
  "last_collection": "2025-10-03T12:15:00Z",
  "next_collection": "2025-10-03T12:30:00Z",
  "status": "healthy",
  "data_points_today": 8
}
```

8.3.3

```
#  (config.py)
ANALYTICS_COLLECTION_INTERVAL = 15  #
ANALYTICS_MAX_DATA_POINTS_PER_DAY = 10  #

# / 0
export ENABLE_BACKGROUND_SERVICES=1  #
export ENABLE_BACKGROUND_SERVICES=0  #
```

8.4

8.4.1

```
#
# : http://localhost:5000/admin/users/create

#
python scripts/create_user.py \
  --email admin@hashinsight.net \
  --username admin \
  --role owner \
  --password-prompt

# :
# : ********
#
# ID: 123
# Email: admin@hashinsight.net
# Role: owner
```

8.4.2

`owner`		
`admin`		
`broker`		
`client`		

```
#
python scripts/change_user_role.py \
  --user-id 123 \
  --new-role admin

#
python scripts/bulk_import_users.py --file users.csv
```

8.4.3

```
#
from decorators import has_permission

@app.route('/api/miners/delete/<int:miner_id>', methods=['DELETE'])
@login_required
@requires_permission('miners:delete')
def delete_miner(miner_id):
    # miners:delete
    pass
```

8.5

8.5.1

HashInsight

```
# config.py
ANALYTICS_MAX_DATA_POINTS_PER_DAY = 10  # 10
```

8.5.2

```
# 90
python scripts/archive_historical_data.py --days 90 --storage s3

# :
#  ...
#  market_analytics: 123,456
#  calculation_history: 45,678
# ⬆   s3://hashinsight-archive/...
#
#  :  5.2 GB
```

8.5.3

```
# 6
python scripts/archive_audit_logs.py --months 6

#  ()
# audit/audit_logger.py 100MB
```

9

9.1 Request Coalescing

HashInsight Request Coalescing **9.8**

```
:
Request 1 → API Call → Response 1
Request 2 → API Call → Response 2
Request 3 → API Call → Response 3
(3API)

Request Coalescing:
Request 1 ┐
Request 2 ├→ Single API Call → Response → Shared Result
Request 3 ┘
(1API67%)
```

```
# cache_manager.py
REQUEST_COALESCING_ENABLED = True
COALESCING_TIMEOUT = 100  # 100ms
COALESCING_MAX_WAIT = 500  # 500ms
```

```
# Request Coalescing
curl http://localhost:5000/api/performance/coalescing-stats | jq
```

```
#
{
  "enabled": true,
  "total_requests": 98234,
  "deduplicated_requests": 85432,
  "api_calls_saved": 85432,
  "performance_improvement": "9.8x",
  "average_wait_time_ms": 45
}
```

9.2

9.2.1

```
--
CREATE INDEX idx_miners_user_id ON miners(user_id);
CREATE INDEX idx_market_data_timestamp ON market_analytics(created_at DESC);
CREATE INDEX idx_calculations_user_created ON calculations(user_id, created_at);

--  ()
CREATE INDEX idx_miners_user_model ON miners(user_id, model);

--  ()
CREATE INDEX idx_active_miners ON miners(user_id) WHERE status = 'active';
```

9.2.2

EXPLAIN

```
EXPLAIN ANALYZE
SELECT m.*, u.username
FROM miners m
JOIN users u ON m.user_id = u.id
WHERE m.status = 'active'
ORDER BY m.created_at DESC
LIMIT 100;

--
```

N+1

```
#   - N+1
miners = Miner.query.filter_by(user_id=user_id).all()
for miner in miners:
    print(miner.user.username)  #
```

```
#   - JOIN
miners = Miner.query.join(User).filter(Miner.user_id == user_id).all()
for miner in miners:
    print(miner.user.username)  #
```

9.2.3

```
# config.py
SQLALCHEMY_ENGINE_OPTIONS = {
    'pool_size': 10,           #  (worker × 2)
    'pool_recycle': 300,       # 5
    'pool_pre_ping': True,     #  (Neon)
    'pool_timeout': 30,        # 30
    'max_overflow': 20,        # 20
    'connect_args': {
        'connect_timeout': 15,         #
        'application_name': 'hashinsight',  #
        'options': '-c statement_timeout=30000'  # 30
    }
}
```

9.3

9.3.1

```
#
L1:  ()
    ↓ miss
L2: Redis ()
    ↓ miss
L3:  ()

#
@cache_manager.cached(ttl=300, level='L1')  # 5
def get_user_profile(user_id):
    return db.session.query(User).get(user_id)

@cache_manager.cached(ttl=3600, level='L2')  # 1
def get_market_data():
    return fetch_from_api()
```

9.3.2 TTL

	TTL	
	5	
	5	
	1	
	10	
	24	

9.3.3

```
#
python scripts/warmup_cache.py

#
# 1.
# 2.
# 3.
# 4.

# :
#  : 1,234
#  : +
#  : 10
#   : 12.3s
```

9.4

9.4.1

```
#   -
for miner in miners:
    daily_revenue = calculate_revenue(
        miner.hashrate,
        btc_price,
        network_difficulty
    )
# : 5000 × 10ms = 50
```

```
#   -
import numpy as np

hashrates = np.array([m.hashrate for m in miners])
revenues = calculate_revenue_vectorized(
    hashrates,
    btc_price,
    network_difficulty
)
# : ~500ms (100)
```

9.4.2

```
from concurrent.futures import ThreadPoolExecutor

#
def process_miner_batch(miners, batch_size=100):
    with ThreadPoolExecutor(max_workers=8) as executor:
        futures = []
        for i in range(0, len(miners), batch_size):
            batch = miners[i:i+batch_size]
            future = executor.submit(process_batch, batch)
            futures.append(future)

        results = [f.result() for f in futures]
    return results
```

9.4.3

```
#  (ORM)
from sqlalchemy import insert

#   -
for data in large_dataset:
    db.session.add(Model(**data))
    db.session.commit()  #

#   -
db.session.bulk_insert_mappings(Model, large_dataset)
db.session.commit()  #

#   - bulk insert
stmt = insert(Model).values(large_dataset)
db.session.execute(stmt)
db.session.commit()
```

9.5

```
#
python scripts/performance_benchmark.py

# :
#  API:
#    - /api/miners: p50=45ms, p95=120ms, p99=250ms
#    - /api/dashboard: p50=80ms, p95=200ms, p99=400ms
#  :
#    - : 15ms
#    -  (>1s): 0
#  :
#    - : 78.5%
#    - Request Coalescing: 9.8x
```

10

10.1 On-Call

09:00-18:00	DevOps		
18:00-09:00			On-Call Manager
/			On-Call Manager

- **PagerDuty**:
- **Slack**: #incident-response
- **Grafana**:
- **Incident.io**:

10.2

P0		15	30	
P1		30	1	API >5s >5%
P2		2	4	
P3		4	8	
P4		1	2	UI bug

10.3

P0/P1

```
1. [0-5]
   ├→ PagerDuty
   ├→ Slack
   └→

2. [5-15]
   ├→  (ACK)
   ├→
   ├→
   └→

3. [15-30]
   ├→
   ├→
   ├→
   └→

4. [30+]
   ├→
   ├→
   ├→
   └→
```

10.4

1:

```
sqlalchemy.exc.TimeoutError: QueuePool limit of size 10 overflow 20 reached
```

```
# 1.
psql $DATABASE_URL -c "
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE state = 'idle'
  AND state_change < NOW() - INTERVAL '10 minutes';
"

# 2.  ()
export SQLALCHEMY_POOL_SIZE=20
export SQLALCHEMY_MAX_OVERFLOW=40

# 3.
systemctl restart hashinsight

# 4.
watch -n 5 "psql $DATABASE_URL -c 'SELECT count(*) FROM pg_stat_activity;'"
```

2: OOM

```
MemoryError: Unable to allocate array
Killed (OOM)
```

```
# 1. worker
kill -HUP $(cat /var/run/gunicorn.pid)

# 2.
redis-cli FLUSHDB

# 3. worker ()
gunicorn --workers 2 --max-requests 500 main:app
```

```
# 4.
watch -n 5 'free -h'
```

3:

- - 401/403 - IP

```
# 1.
export RATE_LIMIT_ENABLED=true
export RATE_LIMIT_REQUESTS_PER_MINUTE=10

# 2. IP ()
ufw deny from 192.168.1.0/24

# 3. Cloudflare ()
# Cloudflare Dashboard -> Security -> DDoS

# 4.
tail -1000 /var/log/gunicorn/access.log | \
  awk '{print $1}' | sort | uniq -c | sort -rn | head -20
```

10.5

```
# 1.
git log --oneline -10

# 2.
git revert HEAD --no-edit
#
git checkout v1.9.5

# 3.
git push origin main

# 4.
curl http://localhost:5000/health | jq '.version'

# 5.
# Slack: "v1.9.5"
```

```
# ⚠ : !

# 1.
python backup/backup_manager.py --type emergency

# 2.
psql $DATABASE_URL < /tmp/backups/hashinsight_backup_20251003_020000.sql

# 3.
python scripts/verify_database_integrity.py

# 4.
systemctl restart hashinsight
```

10.6

```
 [P0 ] HashInsight

: 2025-10-03 14:23 UTC
:
:


30

: #incident-2025-10-03-db-outage
: @john.doe
```

```
 [] HashInsight

: 2025-10-03 14:45 UTC
:
:
:

: 15:00 UTC
```

```
 [] HashInsight

: 2025-10-03 15:12 UTC
: 49
: PostgreSQL
:


24
```

10.7 (Postmortem)

```
# HashInsight  - 2025-10-03

##
- ****: INC-2025-1003-001
- ****: P0
- ****: 2025-10-03 14:23 UTC
- ****: 2025-10-03 15:12 UTC
- ****: 49
- ****: 100% ()

##
- 14:23 - Prometheus:
- 14:25 -
- 14:30 -
- 14:35 -  ()
- 14:45 -
- 15:00 -  ()
- 15:12 -

##
PostgreSQL1030+


##
1. :
2. : 2040
3. : 3

##
- [ ]  (>80%)
- [ ]
- [ ]
```

```
- [ ]

##
```

A:

```
#
gunicorn --bind 0.0.0.0:5000 --workers 4 main:app

#
kill -HUP $(cat /var/run/gunicorn.pid)

#
kill -TERM $(cat /var/run/gunicorn.pid)

#
ps aux | grep gunicorn

#
tail -f /var/log/hashinsight/app.log
```

```
#
psql $DATABASE_URL

#
\dt

#
SELECT count(*) FROM pg_stat_activity;

#
pg_dump $DATABASE_URL > backup.sql
```

```
#
psql $DATABASE_URL < backup.sql
```

```
# Redis
redis-cli

#
redis-cli KEYS 'hashinsight:*'

#
redis-cli FLUSHDB

#
redis-cli INFO memory
```

```
#
curl http://localhost:5000/health | jq

# Prometheus
curl http://localhost:9090/metrics

# SLO
curl http://localhost:5000/api/slo/status | jq
```

B:

.env

```
#
DATABASE_URL=postgresql://user:pass@host:5432/hashinsight_db
SESSION_SECRET=your-secret-key-min-32-chars
ENCRYPTION_PASSWORD=encryption-key-min-32-chars

#
BLOCKCHAIN_ENABLED=true
BLOCKCHAIN_PRIVATE_KEY=0x1234567890abcdef...
BLOCKCHAIN_NETWORK=base-sepolia
BASE_RPC_URL=https://sepolia.base.org

#
BACKUP_DIR=/var/backups/hashinsight
BACKUP_ENCRYPTION_KEY=backup-encryption-key
```

```
BACKUP_RETENTION_DAYS=30
BACKUP_STORAGE_TYPE=s3
BACKUP_STORAGE_BUCKET=hashinsight-backups

# KMS (AWS)
AWS_KMS_KEY_ID=arn:aws:kms:us-east-1:123456789:key/xxxxx
AWS_KMS_REGION=us-east-1
AWS_ACCESS_KEY_ID=AKIAXXXXXXXX
AWS_SECRET_ACCESS_KEY=xxxxxxxxxx

#
ENABLE_BACKGROUND_SERVICES=0  #  1
PROMETHEUS_PORT=9090
SLO_MEASUREMENT_WINDOW=30

# API
COINWARZ_API_KEY=your-coinwarz-api-key
COINGECKO_API_KEY=your-coingecko-api-key
```

systemd

```
# /etc/systemd/system/hashinsight.service

[Unit]
Description=HashInsight Platform
After=network.target postgresql.service

[Service]
Type=notify
User=hashinsight
Group=hashinsight
WorkingDirectory=/opt/hashinsight
EnvironmentFile=/opt/hashinsight/.env

ExecStart=/opt/hashinsight/venv/bin/gunicorn \
  --bind 0.0.0.0:5000 \
  --workers 4 \
  --threads 2 \
  --worker-class gthread \
  --timeout 120 \
  --max-requests 1000 \
  --access-logfile /var/log/hashinsight/access.log \
  --error-logfile /var/log/hashinsight/error.log \
  --pid /var/run/gunicorn.pid \
  main:app

ExecReload=/bin/kill -s HUP $MAINPID
KillMode=mixed
KillSignal=SIGTERM
TimeoutStopSec=30

Restart=on-failure
```

```
RestartSec=10

[Install]
WantedBy=multi-user.target
```

C:

Prometheus

`hashinsight_requests_total`	Counter	
`hashinsight_request_latency_seconds`	Histogram	
`hashinsight_error_rate`	Gauge	
`hashinsight_cache_hit_rate`	Gauge	
`hashinsight_db_query_duration_seconds`	Histogram	
`hashinsight_db_connection_pool_size`	Gauge	
`hashinsight_db_connection_pool_active`	Gauge	
`hashinsight_slo_compliance`	Gauge	SLO
`hashinsight_slo_error_budget_remaining`	Gauge	
`hashinsight_circuit_breaker_state`	Gauge	

SLO

SLO			
	99.95%	<99.9%	<99.5%
P95	≤250ms	>200ms	>300ms
	≤0.1%	>0.5%	>1%
	21.6min/	<20%	<10%

D:

	HTTP		
`AUTH_001`	401		
`AUTH_002`	403		
`AUTH_003`	401	Session	
`DB_001`	500		DATABASE_URL
`DB_002`	500		
`CACHE_001`	503	Redis	Redis
`API_001`	429		
`API_002`	500	API	
`BLOCKCHAIN_001`	500		BLOCKCHAIN_PRIVATE_KEY
`ENCRYPTION_001`	500		ENCRYPTION_PASSWORD

E:

v2.0	2025-10-03	HashInsight Ops Team	
v1.9	2025-09-15	DevOps Team	Request Coalescing
v1.8	2025-08-20	Security Team	KMS mTLS
v1.7	2025-07-10	Platform Team	SLO

: HashInsight Platform Operations Team

:

: ops@hashinsight.net

: https://github.com/hashinsight/operations-manual

: 2025-10-03

: 2026-01-03
