HyperSDK

Day-2 Operations & Maintenance

Operational runbooks for monitoring, scheduling, alerting, troubleshooting, and upgrading HyperSDK in production environments.

Production Ops -- Scheduling -- Troubleshooting -- Upgrades

Operational Tasks

Day-to-day operations: health monitoring, log management, configuration updates, and service upgrades.

Health Monitoring

The daemon exposes health endpoints for liveness and readiness checks. Integrate with systemd watchdog, Kubernetes probes, or external monitoring tools.

- `GET /api/v1/health` -- liveness check
- `GET /api/v1/metrics` -- Prometheus metrics
- systemd watchdog integration via `hypervisord.service`

Log Management

Structured JSON logging with configurable log levels. Logs include request IDs for distributed tracing. Ship to ELK, Loki, or any syslog-compatible target.

- Levels: debug, info, warn, error
- Structured JSON format
- Request ID correlation

Configuration Updates

Config discovery searches `./hypersdk.yaml`, `~/.config/hypersdk/config.yaml`, and `/etc/hypersdk/config.yaml`. Changes require a daemon restart or SIGHUP for hot reload.

- YAML-based configuration

Service Management

Managed via systemd with automatic restart, resource limits, and security hardening. The `hypervisord.service` unit provides production-ready defaults.

- `systemctl status hypervisord`
- `journalctl -u hypervisord -f`

- Environment variable overrides
- Config validation on startup

- Automatic restart on failure

Daily Operations Checklist

Task	Frequency	Method
Check service health	Continuous	Health endpoint + monitoring
Review failed jobs	Daily	`hyperctl list --status=failed`
Check disk space	Daily	Prometheus alerts or df
Review backup success rate	Daily	Dashboard or metrics
Rotate logs	Weekly	logrotate (auto-configured)

Job Scheduling

Recurring backups, dependency chains, retry policies, and time window enforcement.

Recurring Backups

Define recurring backup schedules using cron expressions. The ScheduleManager port handles scheduling, ensuring jobs run at the right time with the right priority.

- Standard 5-field cron syntax
- Timezone-aware execution
- Overlap prevention (skip if previous still running)
- Catch-up policy for missed windows

Dependency Chains

Define job dependencies so tasks execute in order. A migration workflow can chain: snapshot, export, convert, upload, and verify as dependent steps.

- DAG-based dependency resolution
- Automatic failure propagation
- Partial retry (resume from failed step)
- Parallel execution of independent branches

Retry Policies

Configurable retry behavior with exponential backoff. The `CalculateBackoff` function computes delays using the retry policy and attempt count.

- Max retries: configurable (default 3)
- Backoff: exponential with jitter
- Retry-on: specific error types
- Dead-letter queue for exhausted retries

Time Windows

Restrict job execution to defined time windows. Jobs submitted outside windows are queued until the next valid window, protecting production during business hours.

- Window format: "HH:MM-HH:MM"
- Day-of-week restrictions
- Holiday calendars
- Emergency override capability

Schedule Configuration Example

```
hyperctl schedule create \
  --name "nightly-backup" \
  --cron "0 2 * * *" \
  --timezone "America/Chicago" \
  --target "prod-web-*" \
  --type "incremental" \
  --retry-max 3 \
  --retry-backoff "exponential" \
  --window "22:00-06:00"
```

Webhook Notifications

Integrate with Slack, email, and custom endpoints for real-time operational alerts.

Slack Alerts

Rich Slack messages with job status, duration, and provider details. Color-coded attachments: green for success, red for failure. Channel routing by severity.

Email Integration

Send email notifications via SMTP or webhook-to-email services. HTML-formatted summaries with job logs attached. Digest mode for high-volume environments.

Custom Webhooks

POST JSON payloads to any HTTP endpoint. Supports custom headers, HMAC signing, and configurable retry. Build integrations with PagerDuty, OpsGenie, or internal tools.

Webhook Event Payload

```
{
  "event": "job.completed",
  "timestamp": "2026-04-02T14:30:00Z",
  "job": {
    "id": "j-abc123",
    "type": "export",
    "provider": "vsphere",
    "vm_name": "prod-web-01",
    "status": "completed",
    "duration_seconds": 420,
    "output_size_bytes": 53687091200,
    "output_format": "ova"
  },
  "signature": "sha256=..."
}
```

Notification Routing Rules

Event	Severity	Channel	Action
job.failed (3+ retries exhausted)	Critical	#ops-critical + PagerDuty	Page on-call
job.failed (retrying)	Warning	#ops-alerts	Notify team
job.completed	Info	#ops-log	Log only
schedule.missed	Warning	#ops-alerts	Notify team

Troubleshooting

Common issues, log analysis techniques, connectivity checks, and diagnostic commands.

Common Issues

Symptom	Likely Cause
Export stuck at 0%	Provider auth expired or network timeout
429 Too Many Requests	Rate limit exceeded; check rate_limit config
WebSocket disconnect	Proxy timeout; increase proxy read timeout
High memory usage	Too many concurrent jobs; reduce worker pool

Log Analysis

Use request IDs to trace operations across log entries. Filter by level and component for targeted investigation.

```
# Follow logs in real-time journalctl -u hypervisord -f # Filter by request ID journalctl -u hypervisord | grep "req-abc123" # Show only errors journalctl -u hypervisord -p err # Last hour of warnings+ journalctl -u hypervisord -p warning --since "1 hour ago"
```

Diagnostic Commands

Service Status

```
systemctl status hypervisord
hyperctl health curl
localhost:8080/api/v1/health
```

Job Investigation

```
hyperctl list --
status=failed
hyperctl query --
id=j-abc123
hyperctl logs --
id=j-abc123
```

Connectivity Checks

```
hyperctl provider test
vsphere hyperctl
provider list curl -k
https://vcenter:443/sdk
```

> *When troubleshooting, always start with the request ID from the error response. It links the API request to every internal log entry for that operation.*

Upgrades & Maintenance

Deployment procedures, zero-downtime upgrades, rollback strategies, and configuration migration.

1. **Pre-Upgrade Checks**

   Verify no active jobs are running. Check changelog for breaking changes. Back up current configuration and binary.

2. **Build & Deploy**

   Run `make deploy` to build the new binary and deploy it. The Makefile handles building, testing, and installing in one step.

3. **Restart Service**

   Restart via systemd: `systemctl restart hypervisord`. The service drains active connections before stopping (graceful shutdown).

4. **Post-Upgrade Validation**

   Verify health endpoint, check logs for startup errors, confirm provider connectivity, and run a test export job.

5. **Rollback (If Needed)**

   Restore the backed-up binary, revert configuration changes, and restart. Rollback completes in under 2 minutes.

Config Migration

When upgrading between major versions, configuration format may change. HyperSDK validates config on startup and reports any deprecated or renamed fields with suggested fixes.

Zero-Downtime Strategy

For environments requiring zero downtime, deploy behind a load balancer. Drain connections from the old instance, upgrade, and re-add to the pool. Active jobs complete on the old instance.

Operationally Mature

HyperSDK is designed for production from day one. Structured logging, health endpoints, graceful shutdown, and systemd integration make day-2 operations straightforward and predictable.