HyperSDK

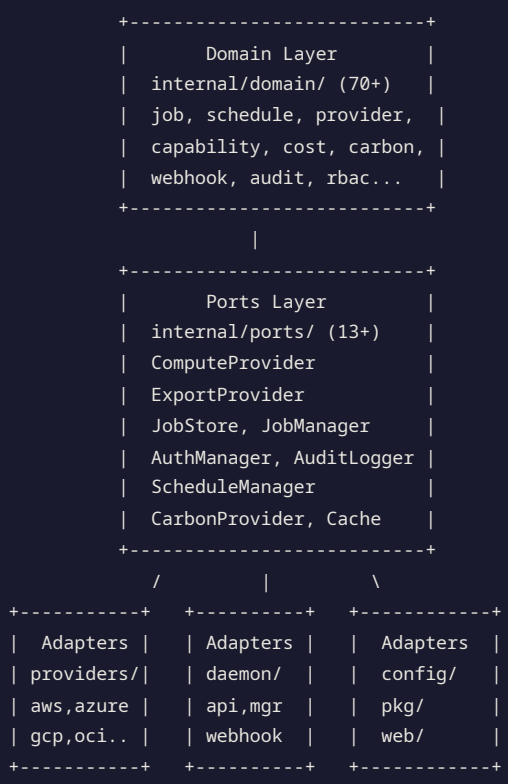
Technical Architecture Deep Dive

Hexagonal architecture, 10-provider plugin system, 150+ API endpoints,
React dashboard, and production-grade security — built in Go 1.24.

Hexagonal Architecture — Domain-Driven Design — Clean Code

Hexagonal Architecture

Clean separation of domain logic from infrastructure concerns.



Domain Types

12 files, 70+ canonical types in internal/domain/. All business logic lives here — jobs, schedules, capabilities, costs, carbon, RBAC, secrets.

Port Interfaces

14 files, 13+ interfaces in internal/ports/. Dependency inversion — domain never imports adapters. Compile-time interface assertions.

Type Aliases

daemon/models/ provides 11 alias files re-exporting domain types. Backward compatibility without breaking adapter imports.

Compile-Time Safety

var _ ports.Interface = (*Concrete)(nil) assertions in every adapter. Broken contracts caught at build time, never at runtime.

Component Overview

8 binaries forming a complete VM management platform.

Binary	Role	Description
hypervisord	Daemon	Central API server with 150+ endpoints, job scheduler, provider management, WebSocket streaming
hyperctl	CLI	Command-line client for provider ops, VM discovery, export jobs, and administration
hyperexport	Export Tool	Standalone VM export utility with format conversion and progress tracking
hyperdash	Dashboard	React + Tailwind web dashboard with VM browser, provider manager, and job tracker
hyperwatch	Monitor	Real-time system metrics, alert management, and health checking
hyperapi	Gateway	API gateway with rate limiting, auth, and request routing
hypersched	Scheduler	Carbon-aware job scheduling with cron support and dependency management
hyperop	Operator	Kubernetes operator for CRD-driven VM migration workflows

Go 1.24
Single Language
Entire Stack

8
Binary
Components

1
make install
Builds Everything

Provider Architecture

ComputeProvider + ExportProvider interfaces with the toVMInfo normalization pattern.

```
// internal/ports/provider.go

type ComputeProvider interface {
    Name() string
    ListVMs(ctx context.Context) ([]domain.VMInfo, error)
    GetVM(ctx context.Context, id string) (*domain.VMInfo, error)
    Capabilities() []domain.Capability
}

type ExportProvider interface {
    ExportVM(ctx context.Context, id string, opts domain.ExportOptions) (*domain.ExportResult, error)
    ExportFormats() []string
}

// Each provider implements toVMInfo() to normalize
// provider-specific data into the canonical domain.VMInfo
```

🔄

toVMInfo Pattern

Every provider converts its native VM representation into domain.VMInfo. Consistent fields: ID, Name, CPUs, MemoryMB, DiskGB, PowerState, OS, Tags.

📦

Provider Registration

Providers register via factory functions. Runtime provider selection by name. Hot-reload supported through the API without daemon restart.

🧪

Test Coverage

Each provider has comprehensive unit tests with mock clients. Interface compliance verified at compile time. Integration tests for API contracts.

📁

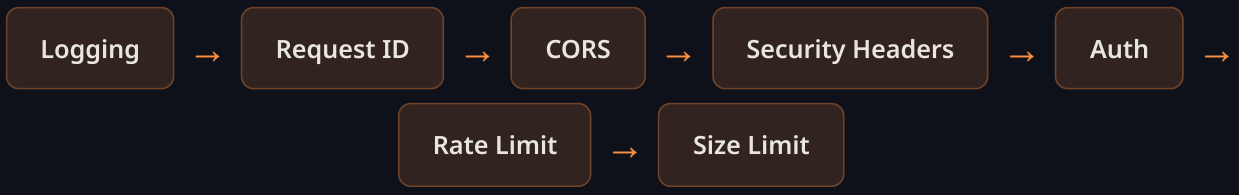
Export Manifests

pkg/common/export_manifest.go provides WriteExportManifest — a generic function used by all providers for consistent export metadata.

API Architecture

150+ endpoints with versioned routes, middleware chain, and WebSocket support.

Middleware Chain



Route Registration

```
// registerVersionedRoute dual-registers at /api/v1/ + legacy paths
registerVersionedRoute(mux, "/providers", s.handleProviders)
registerVersionedRoute(mux, "/vms", s.handleVMs)
registerVersionedRoute(mux, "/jobs", s.handleJobs)
registerVersionedRoute(mux, "/export", s.handleExport)

// All errors via JSONError(w, status, msg)
// Method enforcement via MethodMiddleware(method, handler)
```

🌐

REST + WebSocket

Full REST API for CRUD operations. WebSocket endpoints for real-time job progress streaming and live dashboard updates.

📋

Versioned Routes

All endpoints available at /api/v1/ prefix with legacy compatibility routes. Future versions can coexist without breaking clients.

Web Dashboard

React + Tailwind CSS dashboard with standalone Go-rendered pages.

VM Browser

Discover and browse VMs across all connected providers. Filter by provider, power state, OS. View detailed VM specs.

Provider Manager

Add, configure, and monitor cloud providers. Real-time connection status. Credential management with secure storage.

Jobs Tracker

Monitor export and migration jobs in real-time. Progress bars, logs, retry controls. WebSocket-driven live updates.

Export Workflow

vSphere export wizard with step-by-step guidance. Format selection, CBT options, target configuration.

Alerts & Metrics

System health dashboard with CPU, memory, disk metrics. Alert configuration with notification channels.

Manifest Builder

Visual YAML manifest editor for batch migrations. Template library with common migration patterns.

Standalone Pages (Go-rendered)

`/vms` | `/providers-ui` | `/jobs-ui` | `/export-ui`

Server-rendered HTML pages for lightweight access without React build dependency.

Security Model

Enterprise-grade security built into every layer.

PAM Authentication

Native Linux PAM integration for user authentication. Leverages existing enterprise identity infrastructure — LDAP, AD, SSSD.

Role-Based Access Control

Fine-grained RBAC with roles: admin, operator, viewer. Per-provider permissions. API key and session-based auth.

Rate Limiting

Configurable rate limiting per endpoint and per user. Prevents abuse and ensures fair resource allocation across tenants.

SSRF Protection

All provider URLs validated against SSRF attacks. Private IP ranges blocked. DNS rebinding protection. Path traversal prevention.

Secret Management

SecretManager port interface for provider credentials. Encrypted at rest. Environment variable and file-based secret injection.

Audit Logging

AuditLogger port captures all API operations with user, action, resource, and timestamp. Tamper-evident structured logs.

Deployment Options

From single binary to full Kubernetes deployment.

systemd Service

Production-ready systemd unit file (hypervisord.service). Socket activation, watchdog, resource limits. Standard Linux deployment.

```
sudo systemctl enable hypervisord
sudo systemctl start hypervisord
```

Docker / Podman

Multi-stage Dockerfile for minimal container images. Compose files for full-stack deployment with dashboard and metrics.

```
docker run -p 8080:8080 \
  hypersdk/hypervisord:latest
```

Kubernetes / Helm

Helm chart with CRDs for declarative VM migration. Operator pattern for reconciliation. HPA for auto-scaling.

```
helm install hypersdk \
  charts/hypersdk --namespace vm
```

Architecture Built for Scale

Hexagonal design means every component is replaceable and testable.
From laptop to datacenter — HyperSDK scales with your needs.