# DataPY v1 Capability Commitment & Phased Roadmap

## 1. Overview

This document formalises the capabilities that will be delivered in the first production-ready version (v1) of DataPY. It also lays out a phased development plan so the team can produce a runnable prototype quickly and layer on advanced features incrementally.

## 2. Core Teams and Responsibilities

### Engine (`pype.core.engine.\*`)

* Reads a `.pjob`, launches a local or external Dask client
* Executes components in DAG order, handling data and control edges (`ok`, `error`, `if`, `parallelise`, `synchronise`)
* Performs runtime substitution of `context.\*` variables and `COMP\_\_GLOBAL\_\*` values
* Central retry/timeout logic driven by `job\_config`
* Emits structured events to the Observability layer
* Enforces idempotency flags and skips nodes already checkpointed

### Data Plane (`pype.core.engine.data\_plane`)

* Unified `Pipeline` object that switches between Pandas, Dask, or streaming iterators
* Zero‑copy transfers where feasible
* Memory watermark callbacks so the Engine can throttle fan‑outs
* Extension point for future GPU or Ray back‑ends

### Loader (`pype.core.loader.\*`)

* Jinja‑style templating for `{{context.var}}` and declared globals
* Validation against `job.schema.json` and `component.schema.json`
* Registry look‑ups for component existence and ports
* Guarantees unique component names
* Secret injection via HashiCorp Vault (default provider in v1)

### Planner (`pype.core.planner.\*`)

* Builds a NetworkX DAG with port metadata
* Expands dynamic port groups (e.g. `lookup\*`)
* Detects sub‑job boundaries (Iterator, Joblet)
* Validates multi‑in/multi‑out rules, cycles, dangling ports
* Serialises the graph to `dag.msgpack`

### Registry (`pype.core.registry.sqlite\_backend`)

* Implements the single SQLite table exactly as drafted
* CRUD helpers and CLI commands for registration
* Database migrations stubbed for future component versioning

### Observability (`pype.core.observability.\*`)

* JSON structured logging (one line per event)
* Pluggable metrics exporter interface; Prometheus exporter provided as an OPTIONAL plug‑in
* Event‑hook registry for user‑supplied callbacks (e.g. Slack, OpenTelemetry)

### Checkpointing (`pype.core.checkpointing.\*`)

* Snapshots engine state and globals at sub‑job boundaries to local storage
* Resume logic that skips completed sub‑jobs
* Pluggable remote stores stubbed for v2

### Developer CLI / UX (`pype.cli.\*`)

* `pype build` – creates a `.pjob` artefact
* `pype run` – executes a `.pjob` (with optional context overrides)
* `pype plan` – dry‑run DAG summary
* `pype create component|job` – scaffolds code or YAML templates
* `pype pack` – relocatable bundle for deployment

### Security Utilities (`pype.core.utils.secrets`)

* Vault is the default—and initial—secret provider (environment variables are NOT used)
* Provider interface is pluggable for additional secret back‑ends

### Component Library (`pype.components.\*`)

* `BaseComponent` contract (≤200 LOC)
* Standard components: FileInput, FileOutput, Map, Filter, Iterator, Joblet, Echo
* Dynamic port groups supported (e.g. `lookup\*`, `out\*`)
* Self‑registration into SQLite at install time

## 3. Feature Checklist

* CLI commands: build, run, plan, create component, create job, pack
* YAML structure with job, job\_config, components, connections sections
* Context variable substitution (`{{context.\*}}`)
* Global variable substitution (`COMP\_\_GLOBAL\_\*`)
* Jinja templating and JSON‑Schema validation of job files
* NetworkX DAG construction with multi‑in/multi‑out validation and sub‑job detection
* Local/external Dask execution, retry/timeout handling, idempotency guards
* Automatic switch between Pandas, Dask, and streaming
* JSON logging with optional Prometheus metrics plug‑in
* Local filesystem snapshot and resume
* SQLite component catalogue (table as drafted)
* Delivery of FileInput, FileOutput, Map, Filter, Iterator, Joblet, Echo
* Vault‑based secret injection provider
* ZIP container (`.pjob`) with manifest and msgpack DAG

## 4. Phased Delivery Plan

### Phase 0 – Skeleton & Registry (Days 1‑2)

* Repository scaffold (pre‑commit, CI)
* SQLite registry with CRUD helpers
* BaseComponent and Echo component

### Phase 1 – Loader, Planner, Build Command (Days 3‑4)

* job.schema.json and component.schema.json
* Templating and validation in Loader
* Planner builds basic DAG (no sub‑jobs yet)
* `pype build` produces `.pjob` artefact

### Phase 2 – Engine Runtime (Days 5‑7)

* Data plane (Pandas/Dask switch)
* Engine orchestrator executing sequential DAG
* FileInput and FileOutput components
* `pype run` executes a simple job end‑to‑end

### Phase 3 – Control Flow & Sub‑Jobs

* Iterator and Joblet components
* Multi‑in/out rules and sub‑job detection
* Parallelise/synchronise control edges

### Phase 4 – Observability

* JSON structured logging in production format
* Metrics exporter interface (Prometheus plug‑in optional)
* Event‑hook registry for external notifications

### Phase 5 – Checkpointing

* Local filesystem snapshots at sub‑job boundaries
* Resume logic integrated into Engine

### Phase 6 – Developer UX Polish

* `pype plan` DAG visualiser
* `pype create` scaffolder enhancements
* `pype pack` deployment bundle
* Extended documentation and examples