Table of Contents

1 Technical Competencies 2

1.1 Current Role & Production Systems . . . 2
1.2 Career Background & Experience . . . 2
1.3 Key Production System . . . 2
1.4 Relevant Technical Expertise . . . 2
1.4.1 Python Automation & Workflow Systems . . . 2
1.4.2 Terminal Workflow Standardization . . . 3
1.4.3 Cloud & Container Infrastructure . . . 3
1.4.4 Data Engineering & Analytics . . . 4
1.4.5 System Programming & Performance . . . 4
1.5 Production Examples Relevant to Netstrata . . . 4
1.5.1 Migration Infrastructure Expertise . . . 4
1.5.2 Data Quality & Integrity Systems . . . 5
1.5.3 Internal Automation Tools . . . 5
1.5.4 AI-Augmented Development Tools (Production Context) . . . 5
1.6 Technical Environment & Practices . . . 6
1.6.1 Development Workflow . . . 6
1.6.2 Code Quality Standards . . . 6
1.6.3 Rust Production Experience . . . 7
1.6.4 Rapid Prototyping Capability . . . 7
1.7 What This Means for Netstrata . . . 7
1.7.1 December 2025 Milestone (Complete) + Ongoing Support . . . 7
1.7.2 July 2026 Milestone Contribution . . . 8
1.7.3 Post-2026 Contribution (if still contributing) . . . 8
1.8 Technical Philosophy . . . 8

1 Technical Competencies

1.1 Current Role & Production Systems

Director of Operations at Eon Labs Ltd. (<www.eonlabs.com>), handling administration, HR, SR&ED (Scientific Research and Experimental Development), feature engineering, and AI agent coding tool adoption across the team. Focus areas include Claude Code Agent Skills development—now an industry-standard open format adopted by Claude Code, GitHub Copilot, Cursor, OpenAI Codex, and 5+ other AI coding tools. Developed 18 plugins with 100+ skills in the cc-skills ecosystem (955+ commits since December 2025).

Seeking final career opportunity in team environment at established business, transitioning from independent operations role to hands-on technical contribution.

1.2 Career Background & Experience

25 years of SaaS development experience including founding and scaling internet solutions company (2000) to 100+ employees across Hong Kong, Shanghai, and Macau. Built first-generation web analytics tools (pre-Google Analytics era). Understand both scaling challenges and market positioning importance from operator perspective.

Learned critical lessons about technology adoption patterns, market timing, and the importance of established markets with proven customer bases—which directly informs interest in contributing to Netstrata's established position in NSW strata management with proprietary software at major milestones (December 2025 complete, July 2026 next).

1.3 Key Production System

Operational automation system using Telegram bot interface for team workflow orchestration (quantitative trading infrastructure monitoring):

- **launchd supervision**: Automatic crash detection and restart
- **100ms auto-reload**: Rapid iteration and deployment
- **Doppler credential management**: Secure secrets handling
- **24/7 operation**: Continuous monitoring and maintenance

Demonstrates operational reliability patterns applicable to business-critical systems.

1.4 Relevant Technical Expertise

1.4.1 Python Automation & Workflow Systems

- **Package Management**: uv for modern Python dependency management (avoiding legacy pip/conda)

- **Self-Contained Scripts**: PEP 723 inline dependencies for portable automation tools
- **Production Deployment**: Cloud Run serverless jobs, systemd service management for unattended operation
- **Type Safety**: mypy strict mode, PEP 561 compliance for maintainable codebases
- **Async/Streaming**: Real-time data collection via WebSocket with exponential backoff retry logic
- **API Integration**: Web scraping (Playwright + BeautifulSoup), third-party service orchestration
- **Monitoring**: Dead Man's Switch patterns (Healthchecks.io), mobile alerting (Pushover)

Cost-efficient infrastructure: Free-tier optimization across cloud providers (Cloud Run, BigQuery, managed databases)

Recent Example: Extracted 292 blog posts from Netstrata website using automated web scraping with AJAX pagination handling, rate limiting, and metadata preservation.

1.4.2 Terminal Workflow Standardization

iterm2-scripts - team-scalable terminal workspace automation:

- **Version-controlled workspace configs**: TOML files define tab layouts, pane ratios, and shell commands per squad. New developer clones repo → gets standardized environment automatically
- **Git worktree auto-discovery**: Feature branch worktrees appear as tabs without manual configuration. 3 squads with parallel branches stay organized
- **Shell alias resolution**: Team-wide `.zshrc` aliases resolved at runtime—no hardcoding
- **Persistent preferences**: Tab names, ordering, and selections remembered across sessions with atomic writes
- **Migration support**: Config schema changes handled gracefully as team workflow evolves

Netstrata Application: Squad A (DMS) commits `workspace-dms.toml` with 10 tabs for their module structure. New Manila developer joins → clones repo → starts iTerm2 → selects "DMS" → gets identical environment to senior developers. Setup time: zero. Scales from 8 new hires (Q3 2025) to 15+ (July 2026) without per-person configuration tax.

Technical depth: 5K lines Python, iTerm2 async API, SwiftDialog native UI, 15 modular sources with build-time concatenation (solving AutoLaunch single-file constraint), 4 ADRs documenting design decisions.

1.4.3 Cloud & Container Infrastructure

- **Google Cloud Platform**: Production deployment expertise
  - Free-tier optimization strategies (Cloud Run, Compute Engine, BigQuery)
  - Serverless cron orchestration (Cloud Scheduler)
  - Secret management and zero-downtime credential rotation
- **Container Runtime**: Colima + Docker CLI on macOS for local development and testing
- **Cloud-native architecture**: Dual-pipeline patterns for zero-downtime migrations
- **Cost-efficient SaaS**: Free tier mastery across cloud providers maximizing value without budget bloat

- **Infrastructure as Code**: Configuration management for reproducible deployments

1.4.4 Data Engineering & Analytics

- **Embedded analytics databases**: DuckDB for in-process analytics (10-100x speedups vs traditional approaches)
- **Modern Python data tools**: PyArrow for zero-copy data transfers, async/streaming pipelines
- **Cloud data warehouses**: BigQuery query optimization within free-tier limits
- **Real-time streaming**: WebSocket-based data collection with automatic reconnection
- **Performance engineering**: Window functions, ASOF joins, empirical benchmarking methodologies

1.4.5 System Programming & Performance

- **Rust**: Modern systems programming with cargo toolchain
- **Testing**: cargo nextest for fast test execution
- **Pre-commit Hooks**: Automated code quality checks before deployment

1.5 Production Examples Relevant to Netstrata

1.5.1 Migration Infrastructure Expertise

Dual-pipeline migration architecture enabling zero-downtime system transitions:

- Run legacy and modern systems in parallel with automatic deduplication
- Idempotent operations allowing safe re-runs during failures
- Chunked processing patterns preventing memory failures on large datasets
- Rollback mechanisms for quick recovery from migration issues

Zero-gap data validation frameworks:

- Automated completeness verification across entire datasets
- Database-level constraints (CHECK, PRIMARY KEY) enforcing data integrity
- Structured audit trails providing queryable migration history
- Exception-only error handling preventing silent data corruption

Relevance to Future Migrations: External customer migrations (when needed) require similar dual-pipeline architecture, gap detection, and rollback capabilities. Richardson (WA) is currently stable on existing modules.

1.5.2 Data Quality & Integrity Systems

Multi-layer validation frameworks for production data pipelines:

- HTTP/API layer validation (status codes, rate limits, authentication)
- Schema validation (Pydantic, type checking)
- Sanity checks (database constraints, business rule validation)
- Gap detection (completeness verification across sequences)
- Anomaly detection (outlier identification, statistical validation)

Reliability patterns:

- Atomic file operations preventing partial writes and data corruption
- INSERT OR REPLACE for idempotent database operations
- Structured audit trails stored in compressed formats (Parquet, Arrow)
- Exception-only error handling (no silent failures, no defaults)

Relevance to Core Development: Data integrity assurance applies across scheme records, financial history, and compliance documents—critical for the completed December 2025 DMS release and ongoing Strata Space development toward July 2026.

1.5.3 Internal Automation Tools

Created workflow automation systems that:

- **Eliminate Manual Data Entry**: Bulk operations replacing repetitive form filling
- **API Orchestration**: Coordinating multiple third-party services
- **Report Generation**: Automated document creation from structured data
- **Notification Systems**: Real-time alerts via Telegram/email for operational events

Relevance to Phase 1: Andrew Tunks' operational efficiency mandate aligns with automation capabilities.

1.5.4 AI-Augmented Development Tools (Production Context)

Extensive production experience with AI coding agents, powered by Claude Opus 4.5 (November 2025)—Anthropic's most capable model achieving 80.9% on SWE-bench Verified:

- **Claude Code CLI**: 1000+ hours with Agent Skills development—18 plugins, 100+ skills in cc-skills ecosystem
- **Anthropic's Claude Code (headless mode)**: Programmatic agent orchestration for automated workflows
- **Codex**: OpenAI's code generation model for AI-powered development
- **Cursor**: AI-augmented IDE with context-aware code generation

Agent Skills Industry Standard: Skills are now adopted by 8+ major AI coding tools (Claude Code, GitHub Copilot, Cursor, OpenAI Codex, Mistral Vibe, Kiro, OpenCode). Skills written for one tool work across all—no vendor lock-in. If Tom's team later adopts different AI tools, their skills transfer.

Research Acceleration: AI Research Scraping pattern using Firecrawl + Gemini 3 Pro Deep Research accelerates research on NSW strata legislation changes, competitor documentation, and technical decision-making.

Focus areas: Prompt engineering, context window management, systematic workflow integration, reliability patterns, cost management

NOT speculative AI hype - this is production experience with measurable outcomes (hours saved, errors reduced, processing speed). Detailed AI methodology described in attached document (02).

1.6 Technical Environment & Practices

1.6.1 Development Workflow

- **Unix-Like Systems**: macOS/Linux (POSIX shells, standard conventions)
- **Version Control**: Git with git-town workflow enforcement—blocks raw `git checkout -b`, `git merge`, `git rebase` commands, requiring standardized equivalents. Ensures 8 new developers + 3 squads follow consistent branching patterns from day 1.
- **Testing**: Automated test suites with fast feedback loops
- **Documentation**: Machine-readable specs (OpenAPI, JSON Schema, YAML)
- **Knowledge Capture**: Terminal session recording with 950:1 compression (3.8GB session → 4MB searchable text). Senior developers record debugging sessions for onboarding materials—"watch how I solved X" with full terminal context.
- **Cloud-Native Deployment**: Serverless cron jobs, auto-scaling VMs, managed secret rotation
- **Free-Tier Optimization**: Strategic service selection to minimize operating costs
- **Task Orchestration**: mise hub-spoke architecture—root `mise.toml` defines Python/Node versions; squad folders define domain tasks (`validate:dms`, `test:buildings`). Squads stay synchronized without stepping on each other.

1.6.2 Code Quality Standards

- **Dependency Auditing**: cargo deny check for security vulnerabilities
- **Silent Failure Detection**: itp-hooks catches runtime bugs that fail silently—bare `except:` (hides KeyboardInterrupt), `subprocess` without `check=True` (silent failures). Does NOT flag cosmetic issues (unused imports, PEP8 style)—respects Ted's observation about AI producing "garbage" by focusing only on code correctness.
- **Cross-Module Validation**: Symmetric dogfooding—DMS module validates against Buildings module before release, and vice versa. Catches integration failures between squads before production.
- **Automated Testing**: Continuous validation before deployment
- **Error Handling**: Graceful failure and recovery patterns
- **Observability**: Logging, monitoring, crash detection

1.6.3 Rust Production Experience

rangebar - Published on Crates.io (v2.0.0, 1,863 downloads) - financial data processing library:

- **Workspace Architecture**: Modular crate design with 8 sub-crates for maintainability
- **Algorithm Specification**: Formal mathematical specifications with invariant testing
- **Documentation Management**: Authoritative spec pattern (single source of truth for algorithm behavior)
- **Production Patterns**: Fixed-point arithmetic, non-lookahead bias guarantees, temporal integrity

Documentation Standards:

- Centralized specifications replacing duplicated documentation
- Version-tracked algorithm specs with breaking change history
- Comprehensive testing validating conformance to specifications

Relevance to December 2025 (Complete) + July 2026 Milestones: Software completion requires documentation discipline and specification rigor. Authoritative spec patterns prevent divergence between code, tests, and documentation across Tom Bacani's team (3 squads working in parallel).

Terminology Consistency: Vale glossary management enforces consistent terminology across squads—"lot owner" vs "unit holder", "strata plan" vs "scheme", "by-law" vs "bylaw". Single Source of Truth (SSoT) syncs terminology to all project documentation automatically.

1.6.4 Rapid Prototyping Capability

Speed: MVPs in days, not months **Iteration**: Fast feedback cycles with 100ms reload times **Production Quality**: From prototype to reliable operation quickly

Example: Netstrata blog extraction project went from concept to 292 extracted posts in under 48 hours.

1.7 What This Means for Netstrata

1.7.1 December 2025 Milestone (Complete) + Ongoing Support

DMS, Tasks, Time Recording release delivered - "biggest operational shift since Financials went live" [Tom Q3 Report, p.6]. Post-crunch stabilization underway:

- **Modern Tooling**: Python/uv, Rust for performance-critical components
- **Automation Expertise**: Reducing manual operational overhead
- **Production Reliability**: Experience with 24/7 supervised systems
- **Fast Iteration**: Quick turnaround on features and bug fixes
- **Squad Support**: Experience supporting parallel workstreams

1.7.2 July 2026 Milestone Contribution

Support Buildings, Asset Management, Strata Manager Dashboard - targeting “90-95% of workload in Strata Space” [Tom Q3 Report, p.10]:

- **Data Systems**: Asset management workflow automation
- **Dashboard Development**: Production monitoring interfaces
- **Testing Frameworks**: Automated validation
- **Documentation**: Technical documentation generation

1.7.3 Post-2026 Contribution (if still contributing)

- **Commercialisation Support**: Deployment automation for external customers
- **Automation Enhancement**: Continued workflow optimization
- **Support Infrastructure**: Issue tracking, documentation, monitoring

1.8 Technical Philosophy

Focus on Business Value:

- Automation should save measurable time (hours → minutes)
- Systems should run unattended (launchd supervision, crash recovery)
- Tools should be maintainable (clear code, good documentation)
- Technology serves operations (not technology for its own sake)

Avoid Hype:

- No claims about AI capabilities without production evidence
- Focus on solved problems, not potential future benefits
- Honest about limitations and trade-offs
- Measure outcomes (time saved, errors reduced, uptime)

Prioritize Future-Proof Technologies:

- **Widely-adopted languages**: Python, Rust, C++ (large ecosystems, strong hiring pools, long-term support)
- **Cost-efficient SaaS**: Free-tier mastery, serverless architectures, pay-per-use models
- **Cloud-native patterns**: Idempotent operations, automatic retries, exponential backoff
- **Type safety**: Static type checking (mypy, rustc) preventing runtime errors
- **Popular frameworks**: Established tools with large communities ensure maintainability
- **Open standards**: OpenAPI, JSON Schema, machine-readable specifications

Rationale: Popular technologies = larger talent pool + better tooling + longer support lifecycle. Cost efficiency protects $12-14M software investment from operational bloat.

Join Existing Team:

- Work with Tom Bacani’s software operations team (3 squads)
- Support Andrew Tunks’ operational efficiency mandate
- Contribute to July 2026 milestone (December 2025 complete)
- Team member, not external auditor
- Collaborate with Epi Neto’s IT team (AI initiatives are his domain)