System Prompts v2.0 - REFINED SUMMARY

systemd, HTMX, Ansible, Bash, Python

Project: Next-Generation Hosting Control Panel
Version: 2.0 (Refined)
Date: November 2, 2025

WHAT CHANGED FROM v1.0

Orchestration

v1.0: Docker + Kubernetes
v2.0: systemd (lightweight, battle-tested, direct OS integration)

Why: Simpler to understand, less overhead, easier debugging, better for single-server or small clusters

Frontend

v1.0: React 18 + TypeScript + Redux
v2.0: HTMX (server-side rendering with JavaScript enhancement)

Why: Simpler, minimal JavaScript, faster development, better SEO, server-driven UX

Infrastructure Automation

v1.0: Terraform + Manual scripts
v2.0: Ansible + Bash + Python (idempotent, version-controlled, easy)

Why: Clearer, more maintainable, better for team collaboration

Task Scheduling

v1.0: RabbitMQ + Celery
v2.0: systemd timers (built-in, no external dependencies)

Why: Simpler, fewer moving parts, reliable

COMPLETE TECH STACK v2.0

Backend

- ✓ Rust 1.75+ (latest stable)
- ✓ Actix-web 4.x (web framework)
- ✓ Tokio async runtime
- ✓ PostgreSQL 14+ (database)
- ✓ Redis 7.x (caching)
- ✓ CLI tools for background jobs

Frontend

- ✓ HTMX 1.9.x (dynamic HTML)
- ✓ Tera/Maud templates (Rust-based)
- ✓ TailwindCSS 3.x (styling)
- ✓ Minimal vanilla JavaScript
- ✓ Server-driven UX

Infrastructure & Orchestration

- ✓ systemd (service management)
- ✓ Ansible 2.13+ (automation)
- ✓ Bash 4.4+ (scripts)
- ✓ Python 3.10+ (automation)
- ✓ GitHub Actions (CI/CD)

Monitoring & Observability

- ✓ Prometheus + Grafana (metrics)
- ✓ systemd journal (logging)
- ✓ Node Exporter (system metrics)
- ✓ Custom app metrics (Rust)

Security

- ✓ HashiCorp Vault or env files (secrets)
- ✓ Let's Encrypt (SSL/TLS)
- ✓ ModSecurity (WAF)
- ✓ systemd security hardening
- ✓ Ansible vault (encrypted vars)

KEY ADVANTAGES v2.0

Simplicity

- Minimal dependencies
- Built-in tools (systemd, cron)
- No container complexity
- Direct OS integration

Performance

- Zero container overhead
- Native system monitoring
- Direct process management
- Lower memory footprint

Reliability

- Battle-tested components
- Simpler debugging
- Clear failure modes
- Easy recovery procedures

Maintainability

- Easier to understand
- Less infrastructure code
- Version-controlled automation
- Team-friendly

Cost

- No orchestration overhead
- Fewer tools to manage
- Simpler to self-host
- Works on minimal hardware

CORE DOCUMENTS PACKAGE v2.0

1. ai-dev-system-prompts-v2-refined.md **(Main)**

- 14 comprehensive sections
- Complete standards
- Code examples for each technology
- systemd best practices
- HTMX patterns
- Ansible playbooks
- Bash scripting
- Python automation

2. quick-reference-v2-refined.md **(Cheat Sheet)**

- Printable quick reference
- Prompting templates
- Service checklists
- Code patterns
- Common commands
- Troubleshooting

3. Documentation Index (Navigation)

- How to use the docs
- Quick start paths
- Technology-specific guidance
- Common scenarios

QUICK START PATHS

Path 1: Solo Developer (Same Day)

1. Read main document (2-3 hours)
2. Print quick-reference card
3. Copy Claude context file
4. Start first feature

Time to productive: 4-6 hours

Path 2: Small Team (1 Week)

1. Add to project repository
2. Team reads main document
3. Practice Ansible playbooks
4. Deploy first service
5. Establish monitoring

Time to productive: 1 week

Path 3: Enterprise Team (2-4 Weeks)

1. Add to repository and Wiki
2. Weekly training sessions
3. Setup CI/CD pipeline
4. Deploy staging environment
5. Train all developers
6. Production deployment

Time to productive: 2-4 weeks

WORKING WITH CLAUDE & COPILOT v2.0

For Claude

Paste relevant section + specific question:

```
[Paste systemd section from v2.0 prompts]

I need to create a systemd service for background job processing:
- Service name: hosting-panel-worker
- Runs at: /usr/local/bin/hosting-panel-cli mail-send
- Dependencies: PostgreSQL, Redis
- Restart on failure
- Logging to systemd journal

Generate the service file and health check script.
```

Result: Production-ready service file + documentation

For GitHub Copilot

In VSCode:

1. Create `.copilot/context.md` with v2.0 prompts
2. Write comments describing intent

3. Let Copilot generate code
4. Review against checklist

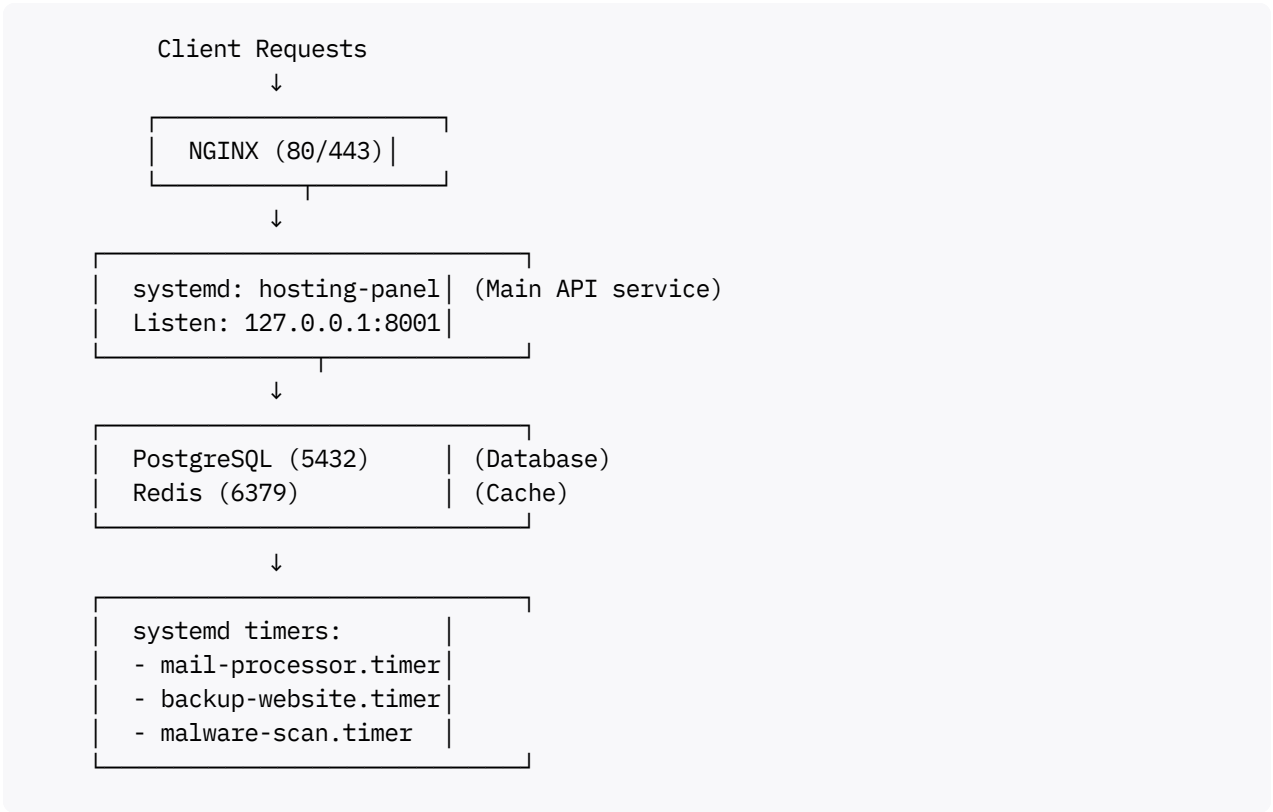
Example:

```
// @workspace Following systemd patterns from .copilot/context.md
// Actix-web service that logs to systemd journal
// Health endpoint returns JSON status
// Graceful shutdown handling

#[actix_web::main]
async fn main() -&gt; std::io::Result&lt;()&gt; {
    // Copilot generates implementation
}
```

ARCHITECTURE AT A GLANCE

Single Server Setup



Deployment Flow

```
Development
    ↓
Git Push (main branch)
    ↓
GitHub Actions
  - Build Rust binary
```

```
  - Run tests
  - Security scan
  - Upload artifact
    ↓
Ansible Deployment
  - Stop service
  - Deploy binary
  - Run migrations
  - Start service
  - Health check
    ↓
Monitoring
  - Prometheus metrics
  - systemd logs
  - Alert rules
    ↓
Production
```

SECURITY FEATURES

Application Level

- Input validation (server-side)
- Output encoding (templates)
- Authentication (Argon2 + JWT)
- Authorization (RBAC)
- SQL injection prevention
- XSS prevention
- CSRF protection
- Rate limiting

Infrastructure Level

- systemd security hardening
- Non-root service user
- ProtectSystem=strict
- ProtectHome=true
- NoNewPrivileges=true
- Capability limiting
- Filesystem restrictions

Deployment Level

- Ansible vault for secrets
- SSH key-based auth
- Version-controlled config
- Idempotent operations
- Rollback capability

COMMANDS YOU NEED TO KNOW

systemd Service Management

```
sudo systemctl start hosting-panel
sudo systemctl stop hosting-panel
sudo systemctl restart hosting-panel
sudo systemctl status hosting-panel
sudo journalctl -u hosting-panel -f
```

Ansible Deployment

```
ansible-playbook playbooks/deploy.yml
ansible-playbook playbooks/deploy.yml --check  # Dry-run
ansible-playbook playbooks/deploy.yml -vvv     # Verbose
```

Build & Test (Local)

```
cargo build --release
cargo test
cargo clippy
cargo fmt
```

View Logs

```
# systemd journal
sudo journalctl -u hosting-panel -n 100
sudo journalctl -u hosting-panel -f  # Follow

# Application logs
tail -f /var/log/hosting-panel/app.log

# Ansible logs
cat ~/.ansible.log
```

WHAT'S INCLUDED IN DOCUMENTS

Main Document (ai-dev-system-prompts-v2-refined.md)

✓ **Section 1:** Development principles (simplicity, security)
✓ **Section 2:** Complete tech stack (Rust, HTMX, systemd, Ansible, Bash, Python)
✓ **Section 3:** Security standards (comprehensive, production-ready)
✓ **Section 4:** Code quality requirements (testing, documentation, style)
✓ **Section 5:** Architecture patterns (systemd services, Ansible playbooks)
✓ **Section 6:** Git workflow (branching, commits, reviews)
✓ **Section 7:** Testing requirements (unit, integration, e2e, security)
✓ **Section 8:** Deployment procedures (Ansible automation)
✓ **Section 9:** API standards (REST, responses, versioning)
✓ **Section 10:** Database standards (PostgreSQL, migrations, optimization)
✓ **Section 11:** HTMX frontend standards (templates, AJAX, forms)
✓ **Section 12:** Infrastructure & sysadmin (systemd, Ansible, monitoring)
✓ **Section 13:** Documentation requirements (code, API, runbooks)
✓ **Section 14:** Code review checklist (security, quality, performance)
✓ **Section 15:** Emergency procedures (service failure, database issues)

SAMPLE: Creating A New Feature

Step 1: Design with Claude

```
[Paste architecture section]

I need to build a user management system with:
- Create/list/update users
- Role-based permissions
- HTMX-based admin UI
- Background verification emails

What should the architecture look like?
```

Claude provides:

- Architecture diagram
- Database schema
- API endpoints
- UI components
- Automation tasks

Step 2: Implement with Copilot

```
// @workspace following architecture from Claude
// Actix-web endpoints for user management
// HTMX forms for admin UI
// Rust CLI for background tasks

#[post("/api/v1/users")]
async fn create_user(...) {
    // Copilot generates
}
```

Step 3: Automate with Ansible

```
# playbooks/deploy-user-management.yml
- name: Deploy user management
  hosts: hosting_servers
  tasks:
    # Ansible generates automated deployment
```

Step 4: Monitor & Verify

```
# systemd manages the service
sudo systemctl status hosting-panel
sudo journalctl -u hosting-panel

# Prometheus collects metrics
# Grafana visualizes
# Alert rules trigger if issues
```

VALIDATION CHECKLIST

Before First Deployment

Code Quality

- [ ] Tests pass locally
- [ ] No compiler warnings
- [ ] Code reviewed
- [ ] Security scan passed

Infrastructure

- [ ] systemd service file ready
- [ ] Ansible playbook tested
- [ ] Monitoring configured

- [ ] Backup procedures in place

Security

- [ ] Secrets not in code
- [ ] Input validation enabled
- [ ] Authorization checks present
- [ ] Encryption configured

Operations

- [ ] Runbooks written
- [ ] Emergency procedures documented
- [ ] Team trained
- [ ] Monitoring dashboards ready

BENEFITS OF v2.0 APPROACH

vs. Docker/Kubernetes

✓ Simpler (no container abstractions)
✓ Faster (no container overhead)
✓ Easier to debug (direct access)
✓ Less infrastructure code
✓ Works on minimal hardware

vs. React

✓ Simpler (server-driven UX)
✓ Faster (less JavaScript)
✓ Better SEO (semantic HTML)
✓ Easier to maintain (no npm hell)
✓ Smaller bundle size

vs. Manual Deployment

✓ Repeatable (Ansible automation)
✓ Faster (one-command deploy)
✓ Safer (pre-tested playbooks)
✓ Version controlled (in Git)
✓ Team friendly (clear process)

NEXT STEPS

This Week

- [ ] Read main document
- [ ] Print quick-reference card
- [ ] Setup GitHub Copilot context
- [ ] Run one example Ansible playbook

This Month

- [ ] Deploy first service with systemd
- [ ] Build first HTMX feature
- [ ] Setup monitoring
- [ ] Train team

This Quarter

- [ ] Full production deployment
- [ ] Monitor metrics
- [ ] Optimize based on results
- [ ] Update documentation

CONTACT & SUPPORT

Technical Lead: [Email]
Questions: Slack #development-standards
Wiki: https://wiki.internal/development
Repository: [Your repo URL]

FINAL THOUGHTS

This v2.0 refined approach prioritizes:

✨ **Simplicity** - Minimal complexity, maximum clarity
 Security - Built-in from start, not bolted on
 Speed - Faster development, deployment, iteration
 Team - Easier to understand, maintain, collaborate
 Cost - Lower infrastructure costs

You now have everything needed for **enterprise-grade, production-ready** hosting control panel development using:

- **Rust backend** (type-safe, performant)

- **HTMX frontend** (simple, effective)
- **systemd orchestration** (reliable, lightweight)
- **Ansible automation** (repeatable, version-controlled)
- **Prometheus monitoring** (observable, alerting)

Build with confidence. Deploy with certainty. Maintain with ease.

Version: 2.0 (Refined)
Date: November 2, 2025
Created by: Development Team
For: Claude AI + GitHub Copilot Assisted Development

You are ready to build enterprise-grade hosting infrastructure!