Cicerone AI Infrastructure Assistant - Technical Paper

CONTENTS

Cicerone AI Infrastructure Assistant - Technical Paper

Abstract

Cicerone is an AI-powered infrastructure management system that provides natural language control over servers, containers, and services. This paper describes the architecture, implementation, and security considerations of the system.

1. Introduction

Managing infrastructure typically requires deep knowledge of command-line tools, SSH access, and manual intervention. Cicerone democratizes infrastructure management by providing a natural language interface that translates user requests into executable commands while maintaining safety through whitelists, permissions, audit logging.

2. System Architecture

2.1 Components

2.2 Data Flow

1. User submits natural language command via web or API
2. Task parser extracts intent and parameters
3. Permission system checks if operation requires approval
4. If approved, task executor runs the command
5. Results logged to database and returned to user
6. Telegram notifications sent for important events

3. Natural Language Processing

3.1 Task Parser

The task parser uses pattern matching to convert natural language into executable commands:

```
        TASKS = {
            'docker_ps': {
                'patterns': [r'show docker containers', r'docke
ps', r'list containers'],
                'category': 'docker',
                'risk': 'low',
                'command': 'docker ps'
            },
            'docker_restart': {
                'patterns': [r'restart (.+)', r'docker restart
+)'],
                'category': 'docker',
                'risk': 'medium',
                'command': 'docker restart {container}',
                'confirm': True
            }
        }
```

3.2 Entity Extraction

Parameters are extracted using regex patterns:

```
        # Extract container name from "restart photos-app"
        for word in message.split():
            if word not in ['restart', 'docker', 'container']:
                params['container'] = word
```

4. Security Model

4.1 Authentication

- SSO via auth.stsgym.com
- Session cookies shared across *.stsgym.com
- SESSION_COOKIE_DOMAIN='.stsgym.com'

- SESSION_COOKIE_SECURE=True
- SESSION_COOKIE_HTTPONLY=True

4.2 Authorization

Path Whitelist: - Read: `/home/wez/`, `/var/log/`, `/etc/nginx/`, `/tmp/` - Write: `/home/wez/`, `/tmp/`

Host Whitelist: - `miner` (the production server) - `trooper1` (the internal network)

Command Whitelist: - `sensors`, `uptime`, `df`, `free`, `docker`, `systemctl`, `ls`, `cat`, `journalctl`

4.3 Permission System

```
        class PermissionManager:
            DANGEROUS_OPERATIONS = ['restart', 'stop', 'start',
'write', 'delete', 'ssh']

            def request_permission(self, user_id, operation,
details):
                # Creates pending request
                # Returns request ID

            def approve_permission(self, request_id):
                # Marks as approved

            def reject_permission(self, request_id):
                # Marks as rejected
```

4.4 Rate Limiting

```
# nginx.conf
limit_req_zone $binary_remote_addr zone=login_limit:10m rate=10
limit_req_zone $binary_remote_addr zone=api_limit:10m rate=30r/
```

4.5 Audit Logging

All commands logged with: - User ID - Command - Parameters - Timestamp - Result

5. Infrastructure Monitoring

5.1 Service State Tracker

```
        class ServiceStateTracker:
            def get_docker_containers(self): ...
            def get_system_services(self): ...
            def get_disk_usage(self): ...
            def get_memory_usage(self): ...
            def get_load_average(self): ...

            def detect_anomalies(self):
                # Returns list of issues:
                # - High disk usage
                # - Low memory
                # - Stopped containers
                # - High load
```

5.2 Error Pattern Recognition

```
        ERROR_PATTERNS = {
            'out_of_memory': {
                'patterns': [r'Out of memory', r'OOM'],
                'severity': 'high',
                'category': 'memory'
            },
            'connection_refused': {
                'patterns': [r'Connection refused', r'Cannot
connect'],
                'severity': 'medium',
                'category': 'network'
            }
        }
```

6. Telegram Integration

6.1 Bot Configuration

```
        class TelegramNotifier:
            def __init__(self):
                self.bot_token = os.environ.get('TELEGRAM_BOT_T
                self.chat_id = os.environ.get('TELEGRAM_CHAT_ID
                self.api_url = f"https://api.telegram.org/
bot{bot_token}"

            def send_alert(self, title, message, severity):
                # Sends formatted message to Telegram
```

6.2 Notification Types

- **Info**: **i** Informational updates
- **Warning**: ⚠ Potential issues
- **Error**: □ Failures
- **Success**: □ Completed operations

7. Scheduled Tasks

7.1 Cron Jobs

Schedule	Task	Purpose
`0 2 * * *`	Daily backup	Backup databases and code
`0 * * * *`	Security check	Monitor security metrics
`0 8 * * *`	Daily report	Send status to Telegram
`*/5 * * * *`	Proactive monitor	Check for anomalies

7.2 Task Manager

```
        class ScheduledTaskManager:
            def add_task(self, name, task_type, schedule, param
                # Add to schedule

            def execute_task(self, task_id):
                # Run the task
```

8. API Integration

8.1 GitLab API

```
        class GitLabAPI:
            def get_projects(self): ...
            def get_branches(self, project_id): ...
            def create_mr(self, project_id, source, target,
title): ...
            def get_pipelines(self, project_id): ...
```

8.2 Cloudflare API

```
        class CloudflareAPI:
            def get_zone(self): ...
            def get_dns_records(self): ...
            def create_dns_record(self, name, type, content): .
            def purge_cache(self): ...
```

9. Performance Considerations

9.1 Caching

- Service state cached for 5 minutes
- Task history stored in JSON files
- Session cookies with 24-hour expiry

9.2 Concurrency

- Gunicorn workers: 2
- Timeout: 60 seconds
- Connection pool for database

10. Deployment

10.1 Docker Configuration

```
        FROM python:3.11-slim
        RUN apt-get update && apt-get install -y docker.io lm-s
        WORKDIR /app
```

```
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
USER root
EXPOSE 5004
CMD ["gunicorn", "--bind", "0.0.0.0:5004", "app:app"]
```

10.2 Environment Variables

Variable	Purpose
SECRET_KEY	Session signing key
DATABASE_URL	PostgreSQL connection
TELEGRAM_BOT_TOKEN	Telegram bot token
TELEGRAM_CHAT_ID	Telegram chat ID

11. Future Work

- [ ] LLM integration for better NLP
- [ ] Workflow chains (multi-step tasks)
- [ ] Parallel task execution
- [ ] Webhook support
- [ ] Cost tracking

12. Conclusion

Cicerone provides a secure, natural language interface for infrastructure management. The combination of whitelists, permissions, audit logging, and Telegram notifications creates a robust system for managing servers, containers, and services while maintaining security and observability.

Authors: STS GYM Infrastructure Team
Date: 2026-03-14
Version: 1.0.0