

# Securaa Custom Utils - High Level Design

## Document Information

- **Service Name:** Securaa Custom Utils Service
- **Version:** 1.0
- **Date:** September 2025
- **Author:** Development Team
- **Related Documents:** [Low Level Design](#)

## Table of Contents

1. [Executive Summary](#)
2. [System Overview](#)
3. [Solution Architecture](#)
4. [Component Architecture](#)
5. [Data Architecture](#)
6. [Security Architecture](#)
7. [Performance Architecture](#)
8. [Deployment Architecture](#)
9. [Process Flows](#)

## Executive Summary

The Securaa Custom Utils Service is a microservice designed to enable users to create, manage, and execute custom utility functions in a secure, multi-tenant environment. The service supports Python-based transformers and scripts,

providing isolated execution environments through containerization while maintaining comprehensive audit trails and performance optimization.

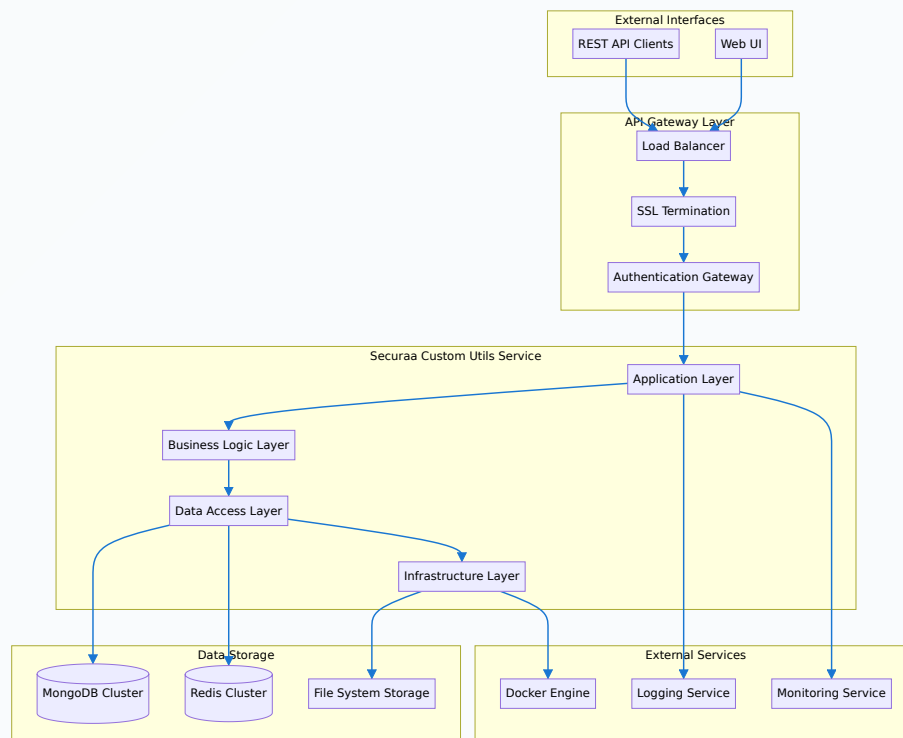
## Key Business Value

- **Extensibility:** Allows users to create custom data processing utilities
- **Security:** Provides secure code validation and isolated execution
- **Multi-tenancy:** Supports multiple tenants with data isolation
- **Scalability:** Designed for high-volume, concurrent operations
- **Compliance:** Comprehensive audit logging for regulatory requirements

## Solution Approach

- **Microservice Architecture:** Self-contained service with clear boundaries
- **Container-based Execution:** Docker containers for secure Python code execution
- **Multi-tier Caching:** Redis and in-memory caching for performance
- **Multi-tenant Data Model:** Isolated data storage per tenant
- **Event-driven Processing:** Asynchronous operations for non-critical tasks

# System Overview



## System Context

The Securaa Custom Utils Service operates within a larger ecosystem of microservices, providing custom utility management capabilities. It integrates with:

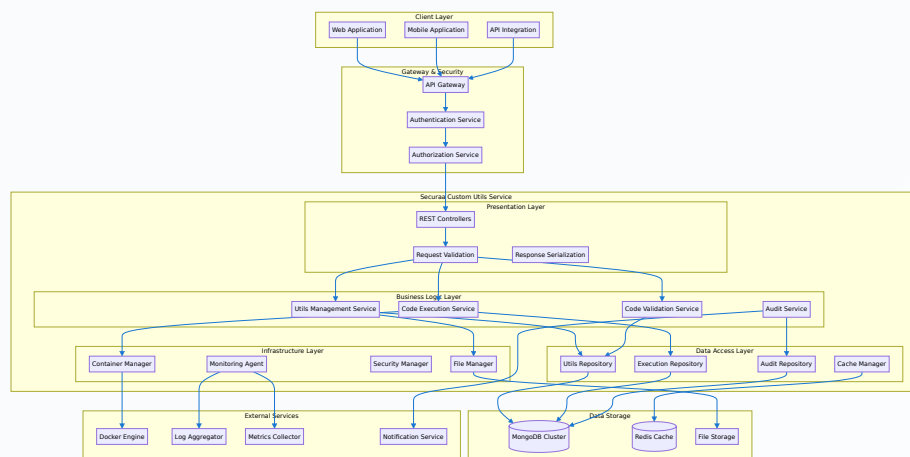
- **Authentication Services:** For user authentication and authorization
- **Tenant Management:** For multi-tenant configuration and isolation
- **Container Orchestration:** For isolated execution environments
- **Monitoring Infrastructure:** For observability and alerting

- **Storage Services:** For persistent data and file management

# Solution Architecture

## Architectural Principles

- **Microservice Architecture:** Independent, deployable, and scalable service
- **Domain-Driven Design:** Clear domain boundaries and responsibility
- **Event-Driven Architecture:** Asynchronous processing for better performance
- **Container-First Design:** Isolated execution environments
- **Multi-Tenant by Design:** Built-in tenant isolation and data security
- **API-First Approach:** RESTful APIs for all interactions



# Component Architecture

## Core Components

### Utils Management Service

- **Purpose:** Manages custom utility lifecycle (CRUD operations)
- **Responsibilities:**
  - Create, read, update, delete utility definitions
  - Validate utility code and metadata
  - Manage utility versioning
  - Handle utility sharing and permissions
- **Key Features:**
  - Multi-tenant utility isolation
  - Version control and rollback capabilities
  - Code validation and syntax checking
  - Dependency management

### Code Execution Service

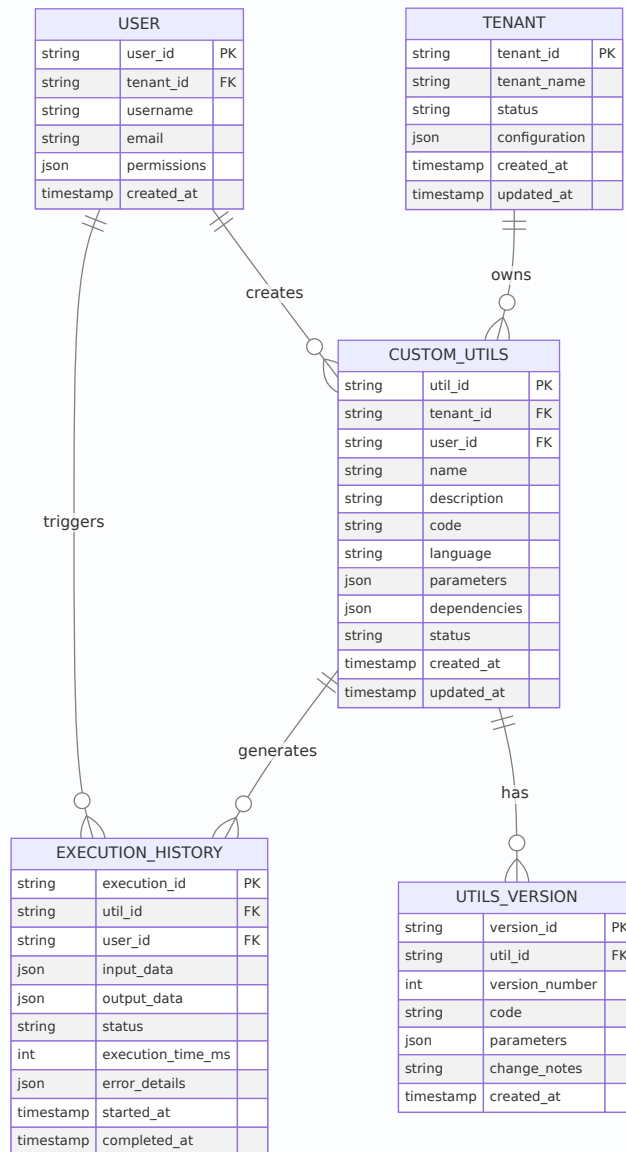
- **Purpose:** Executes utility code in isolated containers
- **Responsibilities:**
  - Create and manage execution containers
  - Execute Python code safely
  - Monitor execution performance
  - Handle execution results and errors
- **Key Features:**
  - Sandboxed execution environment
  - Resource limiting and monitoring
  - Timeout and error handling
  - Parallel execution support

## Validation Service

- **Purpose:** Validates utility code for security and compliance
- **Responsibilities:**
  - Static code analysis
  - Security vulnerability scanning
  - Compliance checking
  - Performance impact assessment
- **Key Features:**
  - AST-based code analysis
  - Blacklisted function detection
  - Resource usage prediction
  - Security rule enforcement

# **Data Architecture**

## **Data Model Overview**



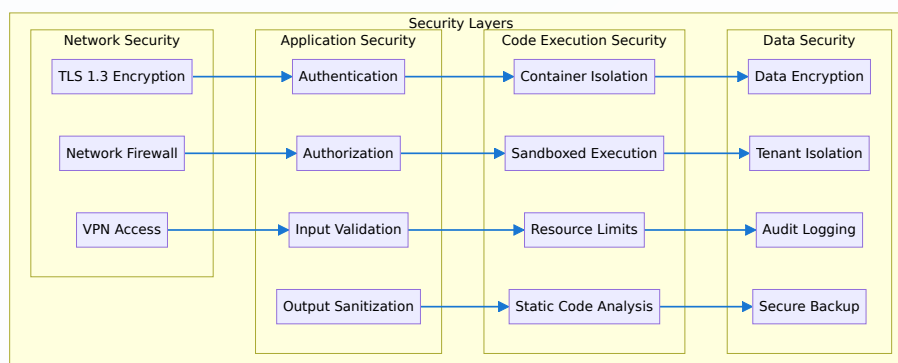


# Data Storage Strategy

- **Primary Data Store:** MongoDB for flexible document storage
- **Caching Layer:** Redis for frequently accessed data
- **File Storage:** File system or object storage for large utility code files
- **Multi-tenancy:** Database-level tenant isolation with tenant-specific collections

# Security Architecture

## Security Layers

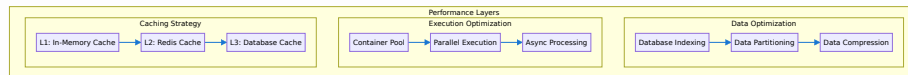


## Code Execution Security

- **Container Isolation:** Each utility execution runs in a separate Docker container
- **Resource Limits:** CPU, memory, and execution time limits enforced
- **Network Isolation:** Containers have no network access by default
- **File System Restrictions:** Read-only file system with limited write access
- **Security Scanning:** Static analysis before execution

# Performance Architecture

## Performance Optimization Strategies

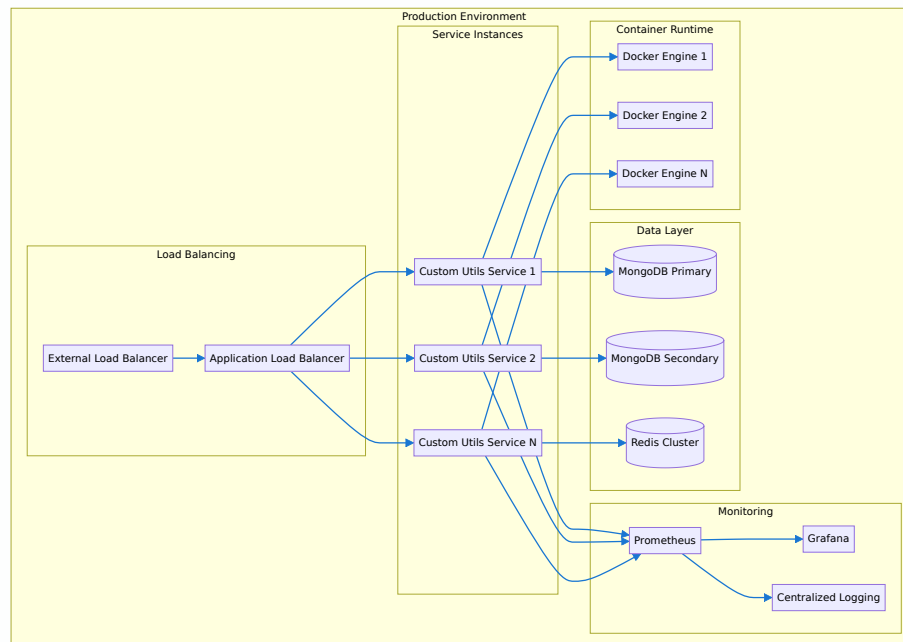


## Scalability Considerations

- **Horizontal Scaling:** Multiple service instances behind load balancer
- **Container Pool Management:** Pre-warmed containers for faster execution
- **Database Sharding:** Tenant-based data distribution
- **Async Processing:** Non-blocking execution for long-running utilities
- **Resource Monitoring:** Real-time performance tracking and alerting

# Deployment Architecture

## Deployment Strategy

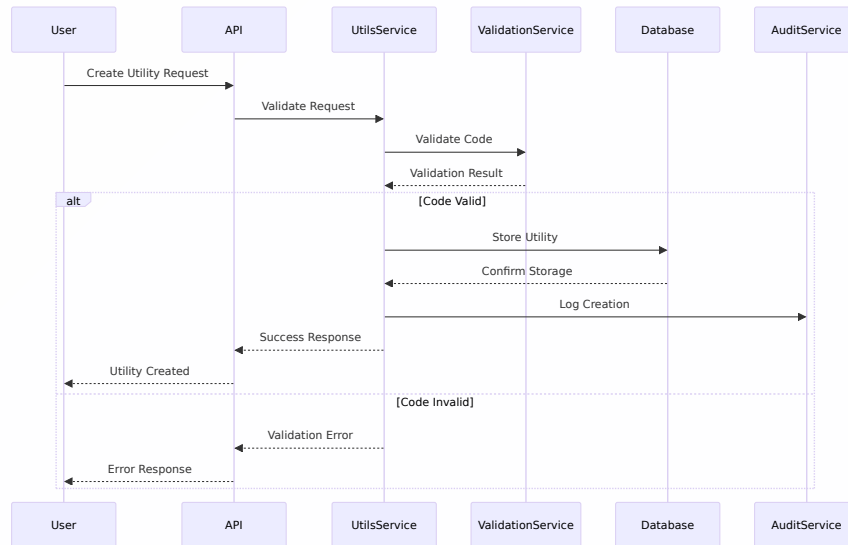


## Infrastructure Requirements

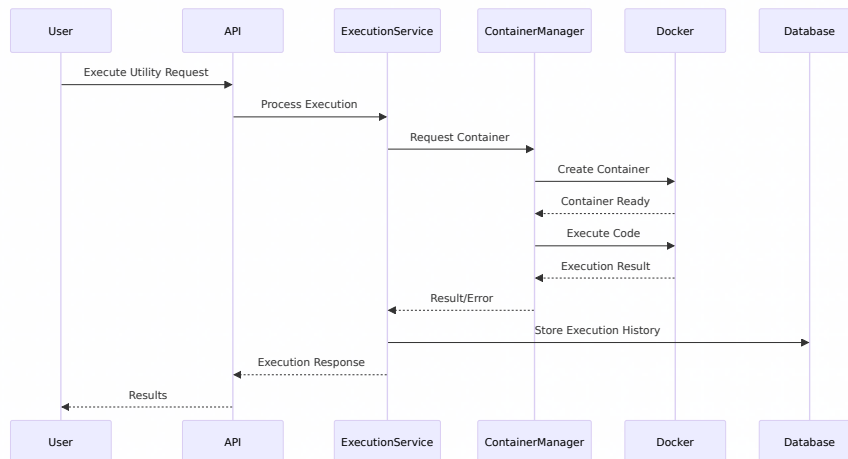
- **Compute Resources:** CPU and memory allocation for service and container execution
- **Storage:** Persistent storage for database and file storage
- **Network:** Low-latency network for inter-service communication
- **Security:** Network segmentation and access controls
- **Monitoring:** Comprehensive observability and alerting

# Process Flows

## Utility Creation Flow



# Utility Execution Flow



## Conclusion

The Securaa Custom Utils Service provides a robust, secure, and scalable platform for custom utility management and execution. The architecture emphasizes security through container isolation, performance through multi-tier caching, and scalability through microservice design patterns.

Key architectural benefits include:

- Secure code execution in isolated environments
- Multi-tenant data isolation and security
- Scalable microservice architecture
- Comprehensive audit and monitoring capabilities
- High availability and fault tolerance