IO-GUARD

Agentic Orchestration for Decentralized GPU Compute

ÖZET:

Bu teknik rapor, otonom yapay zeka ajanları (Auditor, Architect, Sniper) kullanarak dağıtık GPU kaynaklarını yöneten, güvenli ve ölçeklenebilir bir orkestrasyon platformunun mimari detaylarını sunar.

HAZIRLAYAN: **Ali Özen**

TARİH: **24 Ocak 2026**

PROJE LİNKİ : https://github.com/aliozen0/sentinel-io

İçindekiler

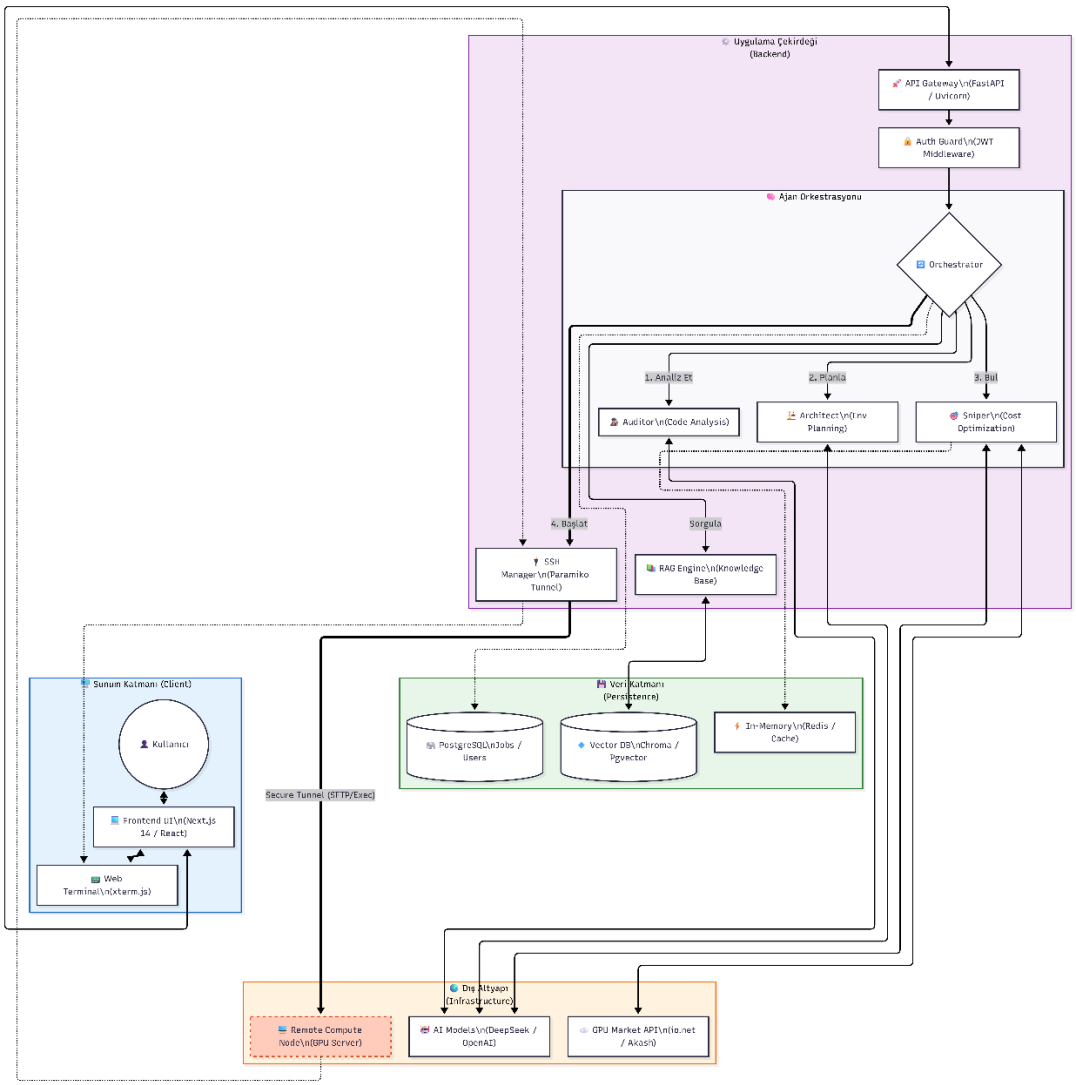
1.Özet 3
2. Teknik Sözlük ve Kısaltmalar 4
3. Sistem Bağlam Diyagramı 4
4. Mimari Tasarım ve Gerekçeler 5
4.1. Backend Motoru: FastAPI ve Asenkron I/O 5
4.2. Frontend: Next.js 14 ve Kompozit UI Mimarisi 6
4.3. Veri Katmanı: Hibrit Kalıcılık ve Vektör Stratejisi 6
4.4. Güvenlik Mimarisi: Ephemeral Credentials 6
5. Kod Tabanı Yapısı 6
5.1. Backend Modülaritesi (The Brain) 7
5.2. Frontend Organizasyonu (The View) 7
6. Konteyner ve Servis Mimarisi 7
7. API Spesifikasyonu ve İletişim Protokolleri 8
7.1. API Tasarım Mimarisi ve Güvenlik Standartları 8
7.2. Çekirdek İş Mantığı ve Ajan Servisleri 9
7.3. Operasyonel Katman ve Veri Servisleri (Operational Layer & Data Services) 10
8. Ajan Orkestrasyon Mantığı 12
8.1. Orkestrasyon Akışı (The Pipeline) 13
8.2. Veri Akış Şeması (Sequence Logic) 13
9. Veri Mimarisi ve Şema Tasarımı 14
9.1. İlişkisel Veri Modeli (Relational Data Model) 14
9.2. Vektör Bellek Mimarisi (Vector Memory Architecture) 14
9.3. Varlık İlişki Diyagramı (ER Diagram) 15
10. Ajan Detayları: Auditor ve Architect 15
10.1. Auditor Ajanı: Kod Sağlığı ve Kaynak Tahmini 15
10.2. Architect Ajanı: Dinamik Ortam Mühendisliği 16
10.3. Ajanlar Arası Veri Kontratı (Data Contract) 16
11. Ajan Detayları: Sniper ve Finansal Orkestrasyon 17
11.1. Arbitraj Stratejisi (The Arbitrage Logic) 17
11.2. Skorlama Algoritması (The Scoring Formula) 17
11.3. Karar Çıktısı ve Veri Yapısı 17
11.4. Bütçe Koruma Mekanizması (Budget Guardrails) 18
12. Ajan Karar Ağacı Şeması (Agent Decision Logic) 18
13. Güvenlik ve Tünelleme Mimarisi (Security & SSH Tunneling Architecture) 20

13.1. Güvenli Tünelleme Protokolü (The Secure Tunnel) 20
13.2. Geçici Kimlik Yönetimi (Ephemeral Credential Handling) 20
13.3. Canlı Veri Akışı (Real-Time Output Piping) 21
14. Kurulum ve Dağıtım Stratejisi (Deployment & DevOps) 21
14.1. Konteyner Orkestrasyonu (Docker Compose) 21
14.2. Akıllı Ortam Algılama (Automatic Environment Detection) 22
14.3. Kritik Ortam Değişkenleri (.env) 22
15. Sınırlılıklar, Riskler ve Gelecek Vizyonu 22
15.1. Mevcut Teknik Sınırlılıklar (Technical Limitations) 23
15.2. Risk Analizi ve Azaltma Stratejileri (Risk Analysis) 23
16. Sonuç ve Değerlendirme 24
17. Kaynakça 25

1.Özet

io-Guard, Merkeziyetsiz Fiziksel Altyapı Ağları (DePIN) üzerinde çalışan makine öğrenimi iş yüklerinin orkestrasyonunu ve maliyet optimizasyonunu sağlamak amacıyla geliştirilmiş, **Ajan Tabanlı (Agentic) bir Katman-2 (Layer-2)** çözümüdür. Küresel GPU arzının parçalı ve heterojen yapısı, yüksek performanslı hesaplama (HPC) kaynaklarına erişimi demokratikleştirse de, bu kaynakların yönetimi ve konfigürasyonu konusunda ciddi operasyonel zorluklar yaratmaktadır. io-Guard, bu karmaşıklığı soyutlayarak, son kullanıcı ile ham donanım arasında otonom bir "Akıllı Katman" (Intelligence Layer) görevi görür.

Sistem, dağıtık ağ sağlayıcıları ile doğrudan entegre çalışan çoklu ajan mimarisi üzerine inşa edilmiştir. Özellikle **Sniper Ajanı**, **io.net** ağındaki global GPU havuzundan gerçek zamanlı piyasa verilerini çekerek "Latency Arbitrage" (Gecikme Arbitrajı) ve maliyet analizi yapar. Bu mimaride; statik kod analizi (Auditor) ve konteynerizasyon planlaması (Architect) gibi kritik DevOps süreçleri, insan müdahalesine gerek kalmadan, olay güdümlü (event-driven) bir yaklaşımla yürütülür. io-Guard, hesaplama gücünü bir meta (commodity) olarak ele alıp, geliştiricilerin altyapı detaylarında boğulmadan sadece algoritmalarına odaklanmasını sağlayan güvenli bir FinOps platformu sunar.



Şekil: io-Guard Platformu Genel Sistem Mimarisi ve Veri Akış Şeması.

2. Teknik Sözlük ve Kısaltmalar

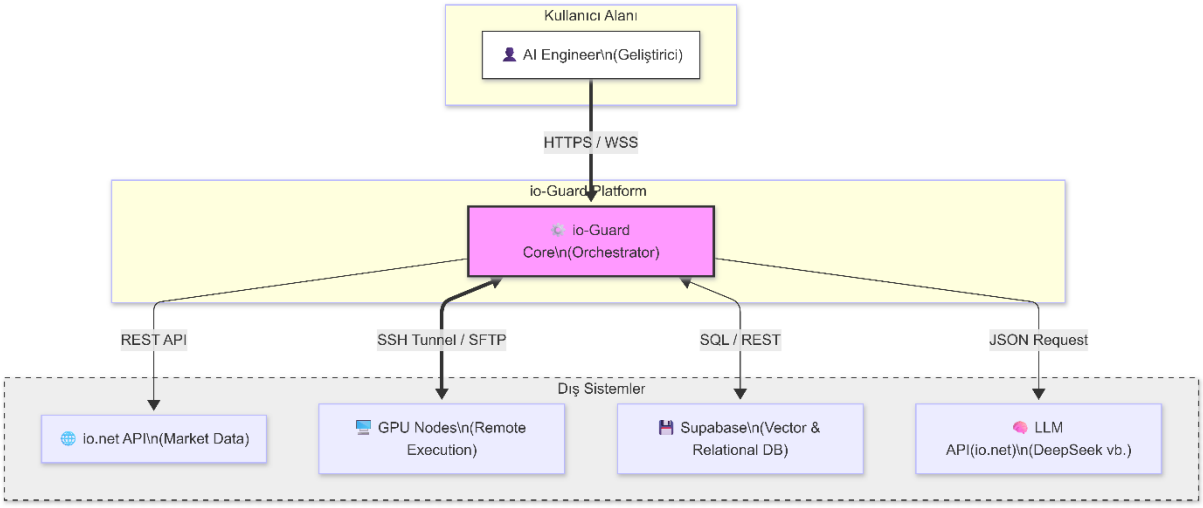
Bu teknik mimari dokümanında kullanılan terminoloji ve kısaltmalar, projenin literatürdeki konumunu ve özgün mühendislik yaklaşımlarını tanımlamak amacıyla aşağıda detaylandırılmıştır:

- **DePIN (Decentralized Physical Infrastructure Networks):** io-Guard'ın üzerinde çalıştığı, blokzincir teknolojisi ile güvence altına alınmış, merkeziyetsiz donanım (GPU/CPU) sağlayıcı ağlarını ifade eden şemsiye terim.
- **Agentic Layer-2 (Ajan Tabanlı Katman-2):** Ham donanım kaynakları ile son kullanıcı uygulaması arasında konumlanan, karar alma mekanizmasının insan müdahalesi olmadan otonom yazılım ajanlarına (Autonomous Agents) devredildiği ara katman mimarisi.
- **RAG (Retrieval-Augmented Generation):** Kullanıcı tarafından yüklenen teknik dokümanların (PDF/TXT) vektörleştirilerek LLM bağlamına (context) dinamik olarak eklenmesini sağlayan, böylece yapay zekanın halüsinasyon oranını düşüren hibrit bellek mimarisi.
- **Ephemeral Credentials (Geçici Kimlik Bilgileri):** Güvenlik prensibi gereği, SSH anahtarları ve sunucu şifreleri gibi hassas verilerin kalıcı disk ünitelerine (HDD/SSD) yazılmadan, sadece işlem süresince RAM (Volatile Memory) üzerinde şifreli olarak tutulması ve işlem bitiminde yok edilmesi prensibi.
- **Hybrid Persistence (Hibrit Kalıcılık):** Sistemin, geliştirme ortamında yerel veritabanı (SQLite) ve üretim ortamında bulut tabanlı veritabanı (Supabase/PostgreSQL) arasında, kod değişikliği gerektirmeden dinamik geçiş yapabilmesini sağlayan veri soyutlama katmanı.
- **Latency Arbitrage (Gecikme Arbitrajı):** Sniper ajanının, farklı coğrafi konumlardaki GPU düğümleri (Nodes) arasındaki ağ gecikmesi ve fiyat farklarını analiz ederek, maliyet/performans oranı en yüksek kaynağı milisaniyeler içinde seçme stratejisi.

3. Sistem Bağlam Diyagramı

Bu bölüm, io-Guard platformunun operasyonel sınırlarını ve dış sistemlerle olan etkileşimini C4 Model (Level 1) standardında görselleştirmektedir. Sistem, kullanıcı (AI Engineer) ile heterojen donanım ve veri sağlayıcıları arasında güvenli ve otonom bir köprü görevi görür.

Kullanıcılar sisteme web arayüzü üzerinden erişirken; io-Guard arka planda GPU pazar yerleri, kimlik doğrulama servisleri ve hesaplama düğümleri ile protokol tabanlı (REST, SSH, SQL) iletişim kurar.



Şekil 1: io-Guard Platformu - C4 Model Seviye 1 Sistem Bağlam Diyagramı.

4. Mimari Tasarım ve Gerekçeler

io-Guard platformu, dağıtık hesaplama kaynaklarının yönetiminde ortaya çıkan yüksek gecikme (latency) ve durum yönetimi (state management) sorunlarını çözmek amacıyla **"Event-Driven Micro-Kernel"** (Olay Güdümlü Mikro-Çekirdek) mimarisi üzerine inşa edilmiştir. Teknoloji yığını (Tech Stack); salt performans değil, aynı zamanda tip güvenliği (type safety), modülerlik ve "Vendor Lock-in" (Satıcıya Bağımlılık) riskini minimize etme prensipleri gözetilerek seçilmiştir.

4.1. Backend Motoru: FastAPI ve Asenkron I/O

Python ekosisteminde Flask veya Django yerine **FastAPI** framework'ünün tercih edilmesinin arkasında üç kritik mühendislik kararı yatmaktadır:

1. **Yüksek Eşzamanlılık (High Concurrency):** io-Guard'ın temel işlevleri olan SSH tünelleme ve LLM çıkarım (inference) süreçleri, doğası gereği "I/O Bound" (Giriş/Çıkış darboğazlı) işlemlerdir. FastAPI'nin `asyncio` ve ASGI (Asynchronous Server Gateway Interface) standardını yerel olarak desteklemesi, tek bir işlemci çekirdeği üzerinde binlerce eşzamanlı WebSocket bağlantısının minimum bellek tüketimiyle yönetilmesini sağlar.
2. **Katı Veri Doğrulama (Strict Data Validation):** `backend/agents/architect.py` ve `backend/models.py` dosyalarında görüldüğü üzere, sistemdeki tüm veri akışı **Pydantic** modelleri ile korunmaktadır. Bu, çalışma zamanı (runtime) hatalarını minimize eder ve ajanlar arasındaki veri kontratlarının (Data Contracts) bozulmamasını garanti altına alır.
3. **Bağımlılık Enjeksiyonu (Dependency Injection):** `get_current_user` ve `get_db` gibi kritik servislerin endpoint'lere enjekte edilmesi, kodun test edilebilirliğini (Testability) artırır ve servisler arası sıkı bağımlılığı (Tight Coupling) engeller.

4.2. Frontend: Next.js 14 ve Kompozit UI Mimarisi

Kullanıcı arayüzü, modern web standartlarını belirleyen **Next.js 14 (App Router)** üzerine kuruludur. Bu katman sadece bir "görünüm" değil, durum yönetiminin (State Management) aktif olarak yapıldığı bir istemci uygulamasıdır.

- **Hibrit Render Stratejisi:** Statik içerikler (Dashboard iskeleti, Dokümantasyon) sunucu tarafında (SSR) oluşturularak ilk yükleme süresi (FCP) optimize edilirken; Terminal emülasyonu ve Canlı Grafik gibi interaktif bileşenler istemci tarafında (CSR) render edilir.
- **Reaktif Veri Akışı:** useCostTracker ve useWizardSession gibi özel kancalar (Custom Hooks), WebSocket üzerinden gelen anlık verileri (GPU sıcaklığı, maliyet sayacı vb.) React state'ine bağlayarak arayüzün yeniden çizilmesini (re-render) tetikler.
- **Atomik Tasarım:** UI bileşenleri, Tailwind CSS ve Radix UI (Shadcn) tabanlı olarak modüler bir yapıda geliştirilmiştir. Bu, tasarım tutarlılığını sağlar ve yeni özelliklerin geliştirme süresini kısaltır.

4.3. Veri Katmanı: Hibrit Kalıcılık ve Vektör Stratejisi

Veri erişim katmanı, uygulamanın çalıştığı ortama göre davranış değiştirebilen polimorfik bir yapıdadır. DBClient sınıfı, "Write Once, Run Anywhere" felsefesini uygular:

- **Veritabanı Agnostik Yapı:** Sistem, yerel geliştirme ortamında hafif **SQLite**, üretim ortamında ise ölçeklenebilir **Supabase (PostgreSQL)** kullanır. Bu geçiş, kod değişikliği gerektirmeden sadece konfigürasyonla sağlanır.
- **RAG ve Vektör Belleği:** Sistemin "Hafıza Çekirdeği" (Memory Core), dokümanları semantik olarak anlamlandırmak için vektör veritabanı teknolojisini kullanır. Yerelde **ChromaDB**, bulutta ise **pgvector** eklentisi kullanılarak, LLM ajanlarına "Uzun Süreli Hafıza" (Long-Term Memory) yeteneği kazandırılmıştır.

4.4. Güvenlik Mimarisi: Ephemeral Credentials

io-Guard, "Security by Design" ilkesi gereği hassas verilerin (SSH Private Keys, Passwords) yönetiminde **Geçici Durum (Ephemeral State)** yaklaşımını benimser.

- **In-Memory Credential Handling:** Kullanıcıların GPU sunucularına bağlanmak için girdiği kimlik bilgileri, JobManager sınıfı içerisinde sadece işlem süresince RAM'de şifreli olarak tutulur. İşlem tamamlandığında veya zaman aşımına uğradığında bu veriler bellekten güvenli bir şekilde silinir (Wipe). Bu sayede, olası bir veritabanı sızıntısında saldırganların sunucu erişim bilgilerine ulaşması imkansız hale getirilir.
- **Tünelleme:** Tüm uzak komut yürütme işlemleri, paramiko kütüphanesi kullanılarak oluşturulan şifreli SSH tünelleri üzerinden gerçekleştirilir, böylece "Man-in-the-Middle" (Ortadaki Adam) saldırı riski ortadan kaldırılır.

5. Kod Tabanı Yapısı

io-Guard projesi, iş mantığı (business logic), veri erişimi ve sunum katmanlarının net sınırlarla ayrıldığı modüler bir mimariye sahiptir. Bu yapı, ekibin farklı modüller üzerinde

birbirini bloklamadan çalışabilmesini (Parallel Development) ve yeni ajanların sisteme kolayca entegre edilebilmesini sağlar.

5.1. Backend Modülaritesi (The Brain)

Backend, "Ajanlar" (Karar vericiler) ve "Servisler" (Uygulayıcılar) olmak üzere iki ana eksende organize edilmiştir:

- **/backend/agents/ (Otonom Karar Mekanizmaları):** Sisteme zeka katan modüllerdir. Her ajan, BaseAgent soyut sınıfından türetilmiştir ve standart bir execute() arayüzünü paylaşır.
  - auditor.py: Statik kod analizi ve bağımlılık tespiti yapar.
  - architect.py: Çalışma ortamını (Docker Image, CUDA sürümü) planlar.
  - sniper.py: io.net üzerinden en uygun GPU kaynağını bulur (Latency Arbitrage).
  - recovery_engine.py: Hata durumunda (örn. SSH kopması) devreye giren "Self-Healing" modülüdür.
- **/backend/services/ (Altyapı ve Araçlar):** Ajanların kararlarını uygulayan "Stateless" yardımcı sınıflardır.
  - orchestrator.py: Analiz hattını (Pipeline) yöneten ve ajanları sırasıyla tetikleyen merkezi orkestratör.
  - ssh_manager.py: Paramiko kütüphanesi ile güvenli tünel açma ve SFTP dosya transferi işlemlerini yönetir.
  - memory_core.py: RAG mimarisi için vektör veritabanı (ChromaDB/Supabase) işlemlerini soyutlar.
- **/backend/routes/ (API Arayüzü):** Dış dünya ile iletişimi sağlayan REST ve WebSocket uç noktalarıdır. İş mantığı barındırmaz, sadece gelen isteği ilgili servise yönlendirir (Routing).

5.2. Frontend Organizasyonu (The View)

Next.js 14 "App Router" yapısına uygun olarak hiyerarşik bir düzende kurgulanmıştır:

- **/app/ (Sayfa Yönlendirmeleri):** URL yollarına karşılık gelen sayfa bileşenleri. deploy, analyze, chat gibi modüller ayrı klasörlerde izole edilmiştir.
- **/hooks/ (Custom Logic):** UI bileşenlerini "kirletmemek" için karmaşık iş mantıkları (WebSocket yönetimi, sayaçlar) buraya taşınmıştır. Örn: useCostTracker.ts.
- **/components/ (UI Kit):** wizard, ui (atomik bileşenler) ve layout şeklinde gruplandırılmış, yeniden kullanılabilir arayüz parçalarıdır.

6. Konteyner ve Servis Mimarisi

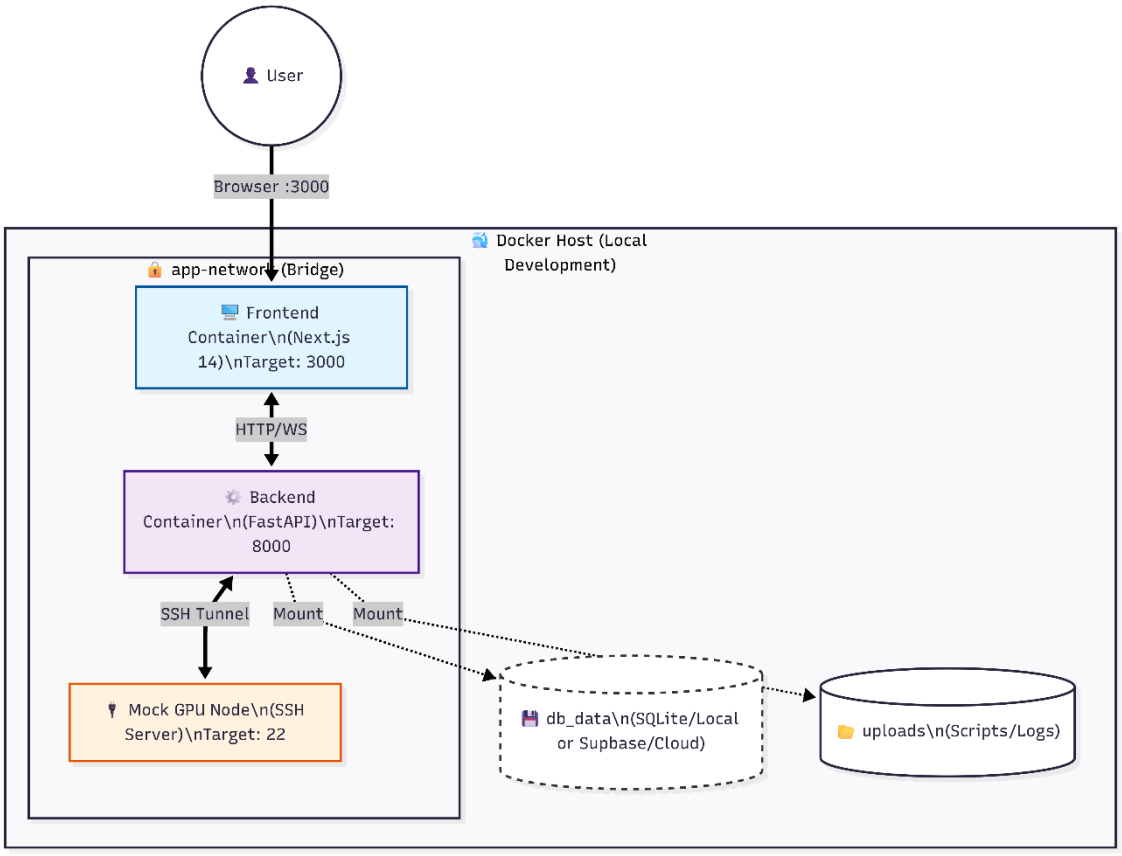
io-Guard platformu, "Infrastructure as Code" (IaC) prensibi doğrultusunda tamamen konteynerize edilmiş bir yapıda dağıtılmaktadır. Geliştirme, test ve üretim ortamları arasındaki "Configuration Drift" (Konfigürasyon Sapması) riskini ortadan kaldırmak amacıyla tüm servisler **Docker** ekosistemi üzerinde izole edilmiştir.

Sistemin orkestrasyonu, docker-compose.yml dosyası üzerinden yönetilen çoklu konteyner mimarisine dayanır. Bu yapı, servislerin birbirleriyle güvenli bir iç ağ (Bridge

Network) üzerinden haberleşmesini sağlarken, dış dünyaya sadece gerekli portları (3000 ve 8000) açarak saldırı yüzeyini minimize eder.

Temel konteyner birimleri şunlardır:

1. **Frontend (UI):** Node.js runtime üzerinde çalışan Next.js uygulamasıdır. Kullanıcı trafiğini karşılar.
2. **Backend (Core):** Python 3.10+ tabanlı FastAPI servisi. İş mantığını ve ajanları barındırır.
3. **Mock GPU Node:** Geliştirme sürecinde gerçek sunucu maliyeti yaratmamak için SSH protokolünü simüle eden sanal Linux ortamıdır.



Şekil 2: *io-Guard Platformu Konteyner İletişim ve Veri Akış Şeması (Docker Host).*

7. API Spesifikasyonu ve İletişim Protokolleri

7.1. API Tasarım Mimarisi ve Güvenlik Standartları

io-Guard platformunun iletişim omurgası, durumsuz (stateless) operasyonlar için **RESTful HTTP**, olay güdümlü (event-driven) veri akışları için ise **WebSocket (WSS)** protokollerini hibrit bir yapıda kullanan "Micro-kernel" mimarisi üzerine kuruludur. Tüm uç noktalar (endpoints), **OpenAPI 3.1.0** spesifikasyonuna uygun olarak belgelenmiş olup, veri serileştirmesi için katı tip denetimli (strongly-typed) JSON şemaları kullanılmaktadır.

Temel Tasarım Prensipleri:

1. **Kaynak Odaklılık (Resource-Oriented):** URL yapısı, eylemlerden ziyade kaynakları temsil edecek şekilde hiyerarşik düzenlenmiştir (örn. /v1/knowledge, /v1/deploy). HTTP fiilleri (GET, POST, DELETE) semantik anlamlarına uygun olarak kullanılır.
2. **Hibrit İletişim:** Kısa süreli istekler (Request/Response) için HTTP/1.1, uzun süreli ve çift yönlü veri akışları (Full-Duplex) için WebSocket protokolü tercih edilmiştir. Bu ayrım, sunucu kaynaklarının optimum kullanımını sağlar.
3. **Güvenlik ve Kimlik Doğrulama (Security First):**
    - **JWT Yetkilendirmesi:** Sunucu tarafında oturum (session) tutulmaz. Tüm korumalı REST uç noktaları, Authorization: Bearer <token> başlığı ile taşınan **JSON Web Token (JWT)** ile doğrulanır. Token yönetimi ve hashleme işlemleri backend/auth.py modülü tarafından yürütülür.
    - **Geçici Kimlik Bilgileri (Ephemeral Credentials):** "Zero-Trust" (Sıfır Güven) ilkesi gereği, SSH anahtarları ve sunucu şifreleri veritabanına **asla** kaydedilmez. /v1/deploy gibi hassas veri içeren isteklerde, bu bilgiler şifreli olarak alınır, yalnızca işlem süresince RAM (In-Memory) üzerinde tutulur ve işlem bitiminde güvenli bir şekilde silinir (Secure Wipe).

Kimlik Doğrulama Servisleri (Auth Domain):

Aşağıdaki tablo, sisteme güvenli giriş ve yetki kontrolünü sağlayan temel servisleri tanımlar:

Metot	Uç Nokta (Endpoint)	İstek Şeması (Payload)	İşlev ve Teknik Detay
POST	/v1/auth/login	Body_login: { "username": "str", "password": "str" }	**OAuth2 Password Flow** standardını uygular. Kullanıcıyı doğrular (bcrypt verify) ve süreli bir access_token döner.
GET	/v1/auth/me	-	Mevcut JWT token sahibinin profil bilgilerini ve güncel kredi bakiyesini döndürür. İstemci tarafındaki oturum durumu (User Context) bu endpoint ile senkronize edilir.

7.2. Çekirdek İş Mantığı ve Ajan Servisleri

Bu bölüm, sistemin otonom karar mekanizmalarını (Analiz, Sohbet, Piyasa) yöneten ve "Agentic" yetenekleri dış dünyaya açan servislerin tam envanteridir. İşlemler genellikle uzun sürdüğü için, bu servisler durum takibi gerektiren (Stateful) süreçleri tetikler.

A. Analiz ve Orkestrasyon Servisleri (Analysis & Orchestration)

Kullanıcı kodunu işleyerek sistemin üçlü ajan yapısını (Auditor -> Architect -> Sniper) tetikleyen uç noktalardır.

Tablo 2: Analiz API Envanteri

Metot	Uç Nokta (Endpoint)	Temel İşlev	Kritik Parametreler (Body/Query)
POST	/v1/analyze	**Kod Analizini Başlat.** Arka planda AnalysisJob başlatır ve takip için job_id döner. Kredi kontrolü yapar.	code (string), budget (float), model (string: "DeepSeek-V3")
GET	/v1/analyze/history	**Geçmiş Analizler.** Kullanıcının daha önceki analiz raporlarını ve durumlarını listeler.	-
GET	/v1/models	**Model Listesi.** Analiz ve sohbet için kullanılabilecek aktif LLM modellerini getirir.	-

B. Etkileşimli Ajan Arayüzleri (Interactive Agent Interfaces)

Kullanıcının doğal dil (Natural Language) kullanarak sistemle veya proje koduyla konuşmasını sağlayan servislerdir.

Tablo 3: Ajan API Envanteri

Metot	Uç Nokta	Ajan Tipi	Yetenek ve İşlev
POST	/v1/chat	**Chat Agent** (Asistan)	**Context-Aware:** Proje koduna ve geçmiş analizlere hakimdir. Kod refactoring veya açıklama yapar.
POST	/v1/chat/ops	**Ops Agent** (Operasyonel)	**Tool Use:** Sistem üzerinde aksiyon alabilir. "Bakiyem kaç?", "Analizi durdur", "Sunucu ne durumda?" gibi komutları çalıştırır.

C. Piyasa İstihbarat Servisleri (Market Intelligence)

Dış kaynaklardan (io.net) veri toplayarak maliyet/performans optimizasyonu yapan servis grubudur.

Tablo 4: Piyasa API Envanteri

Metot	Uç Nokta	Temel İşlev	Veri Kaynağı
GET	/v1/market/status	**Canlı Piyasa Durumu.** GPU stoklarını, fiyatlarını ve io-Guard güvenilirlik skorlarını önbellekten (cache) getirir.	Sniper Agent -> io.net API (In-Memory Cache)

7.3. Operasyonel Katman ve Veri Servisleri (Operational Layer & Data Services)

Bu katman, io-Guard platformunun dış dünya ile fiziksel etkileşime girdiği (SSH/SFTP), dosya sistemini yönettiği ve yapay zeka hafızasını (RAG) yapılandırdığı servis grubudur. Buradaki işlemler genellikle I/O yoğunlukludur ve güvenlik protokollerine (Ephemeral Credentials) sıkı sıkıya bağlıdır.

A. Dağıtım ve Yürütme Motoru (Deployment & Execution Engine)

Uzak GPU sunucularıyla (Remote Compute Nodes) güvenli tünel (Tunneling) kuran ve kod transferini yöneten "Action" servisleridir.

Tablo 5: Dağıtım API Envanteri

Metot	Uç Nokta (Endpoint)	Temel İşlev	Kritik Parametreler (Body)	Teknik Detay
POST	/v1/deploy/execute	**Tekil Script Çalıştır.** .py dosyasını sunucuya yükler ve çalıştırır.	hostname, username, private_key, script_path	Paramiko ile SFTP upload yapar, ardından SSH exec_command ile çalıştırır. Kimlik bilgileri RAM'de tutulur.
POST	/v1/deploy/project	**Proje Dağıtımı.** ZIP veya çoklu dosyaları taşır.	project_dir, entry_point, install_requirements (bool)	Hedef sunucuda proje klasörü yaratır, dosyaları açar ve requirements.txt varsa bağımlılıkları yükler.
POST	/v1/connection/test	**Bağlantı Testi.** SSH kimlik bilgilerini "Dry-Run" test eder.	auth_type ("key"|"password"), passphrase	Sunucuya ping atar ve SSH el sıkışmasını (Handshake) dener, komut çalıştırmaz.
GET	/v1/connections/demo	**Demo Credentials.** Test için sanal GPU (Mock Node) bilgilerini getirir.	-	Kullanıcının sistemi denemesi için yerel Docker konteynerine (Mock SSH Server) ait erişim bilgilerini döner.
POST	/v1/deploy/simulate	**Simülasyon.** Gerçek sunucuya gitmeden dağıtım sürecini taklit eder.	job_config	Arayüz testleri ve eğitim modları için kullanılır.

B. RAG ve Bilgi Yönetimi Servisleri (Knowledge Base & RAG)

Kullanıcıya ait teknik dokümanları (PDF, TXT) vektörleştirerek (Embedding) LLM için "Uzun Süreli Hafıza" oluşturan servislerdir.

Tablo 6: Bilgi Yönetimi API Envanteri

Metot	Uç Nokta	Temel İşlev	Veri Tipi	İşlem Süreci
POST	/v1/knowledge/upload	**Doküman Yükle.**	Multipart/Form-Data (PDF, TXT, MD)	Dosya -> Metin Parçalama (Chunking: 500char) ->

Metot	Uç Nokta	Temel İşlev	Veri Tipi	İşlem Süreci
				Embedding (all-MiniLM-L6-v2) -> Vektör DB (Chroma/Pgvector).
POST	/v1/knowledge/search	**Semantik Arama.**	JSON: { "query": "...", "top_k": 5 }	Kullanıcı sorgusunu vektöre çevirir ve Cosine Similarity ile en yakın doküman parçalarını getirir.
GET	/v1/knowledge/documents	**Doküman Listesi.**	-	Yüklenmiş dosyaların meta verilerini (Dosya adı, ID, Tarih) listeler.
DELETE	/v1/knowledge/{doc_id}	**Doküman Sil.**	Path Param: doc_id	İlgili dokümanı ve ona ait tüm vektör parçalarını veritabanından temizler.
GET	/v1/knowledge/stats	**DB İstatistikleri.**	-	Toplam doküman sayısı ve vektör parçası (chunk) sayısını raporlar.

C. Sistem ve Yardımcı Servisler (System Utilities)

Platformun sağlık durumunu izleyen, geçici dosya yönetimini sağlayan ve dashboard verilerini besleyen altyapı servisleridir.

Tablo 7: Yardımcı Servisler Envanteri

Metot	Uç Nokta	İşlev	Açıklama
GET	/v1/system/status	**Sistem Modu.**	Çalışma ortamını (CLOUD vs LOCAL) bildirir. Frontend özellik setini buna göre ayarlar.
POST	/v1/upload	**Geçici Dosya.**	Yürütme öncesi script dosyalarını sunucu tarafındaki /backend/uploads dizinine yazar.
GET	/v1/uploads	**Dosya Listesi.**	Sunucuda bekleyen (staging) dosyaları ve projeleri listeler.
GET	/v1/dashboard/stats	**Dashboard Metrikleri.**	Ajan durumu, kullanıcı bakiyesi, aktif job sayısı ve sistem sağlık skorunu özetler.

8. Ajan Orkestrasyon Mantığı

io-Guard platformunun temel zekası, sıralı (sequential) ve duruma bağlı (state-dependent) bir karar pipeline'ı üzerine kuruludur. Bu süreç, kullanıcıdan gelen ham kodun analiz edilmesinden, en uygun donanımın kiralanmasına kadar geçen süreci otonom olarak yönetir. AgentOrchestrator servisi, bu akışın merkezi yöneticisidir ve ajanlar arasındaki veri bütünlüğünü sağlar.Süreç, deterministik kurallar ile olasılıksal (probabilistic) yapay zeka çıkarımlarının hibrit bir kombinasyonunu kullanır.

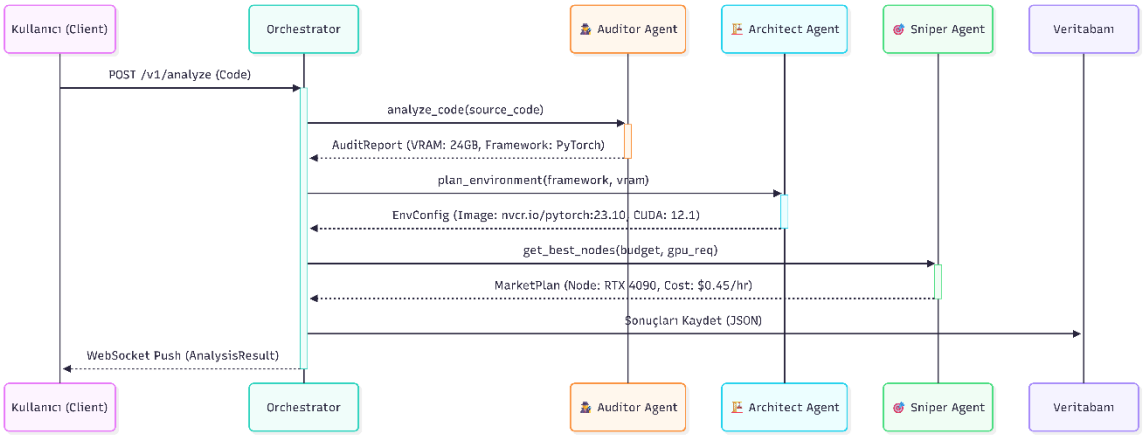
8.1. Orkestrasyon Akışı (The Pipeline)

Bir analiz isteği (POST /v1/analyze) geldiğinde, sistem aşağıdaki 3 aşamalı "Waterfall" sürecini başlatır:

1. **Aşama 1: Kod Sağlığı ve Kaynak Tespiti (The Auditor)**
    - **Girdi:** Ham Python kodu.
    - **İşlem:** Auditor ajanı, kodu hem statik analiz (AST) hem de LLM tabanlı semantik analizden geçirir.
    - **Çıktı:** Tespit edilen framework (PyTorch/TensorFlow), tahmini VRAM gereksinimi (örn. 16GB) ve potansiyel kod hataları (Syntax/Logic Errors).
2. **Aşama 2: Ortam Mimarisi (The Architect)**
    - **Girdi:** Auditor'dan gelen framework ve vram_gb verisi.
    - **İşlem:** Architect ajanı, kodun çalışacağı Docker imajını (Base Image), gerekli Python kütüphanelerini (requirements.txt) ve sistem bağımlılıklarını (apt-get packages) belirler. CUDA sürüm uyumluluğunu (Matrix Compatibility) kontrol eder.
    - **Çıktı:** EnvironmentConfig nesnesi (Tam Dockerfile tarifi).
3. **Aşama 3: Piyasa Arbitrajı (The Sniper)**
    - **Girdi:** Architect'ten gelen donanım gereksinimleri ve kullanıcının budget (bütçe) parametresi.
    - **İşlem:** Sniper ajanı, io.net ağındaki binlerce GPU düğümünü tarar. (Fiyat / Performans) * Güvenilirlik formülü ile bir skorlama yapar ve en optimal düğümü seçer.
    - **Çıktı:** Kiralama için hazır node_id ve maliyet raporu.

8.2. Veri Akış Şeması (Sequence Logic)

Aşağıdaki diyagram, bu ajanların birbiriyle nasıl paslaştığını ve veri yapılarını nasıl dönüştürdüğünü göstermektedir.



Şekil 3: io-Guard Ajan Orkestrasyon ve Veri Dönüşüm Akışı (Sequence Diagram).

8.3. Hata Yönetimi ve Geri Alma (Error Handling & Rollback)

Orkestratör, "Fail-Fast" prensibiyle çalışır. Eğer Auditor kodda kritik bir sözdizimi hatası bulursa, süreç Architect aşamasına geçmeden durdurulur ve kullanıcıya detaylı hata raporu döner. Benzer şekilde, Sniper bütçeye uygun GPU bulamazsa işlem iptal edilir ve kullanıcıya "Bütçe Artırımı" önerilir.

9. Veri Mimarisi ve Şema Tasarımı

io-Guard platformu, yapısal veriler (kullanıcılar, işler) için ilişkisel veritabanı modelini, yapısal olmayan anlamsal veriler (RAG dokümanları) için ise vektör gömme (Vector Embedding) teknolojilerini bir arada kullanan **Poliglot Kalıcılık (Polyglot Persistence)** mimarisine sahiptir.

Veri erişim katmanı, geliştirme ortamında hafiflik sağlamak için yerel **SQLite** dosyasını, üretim ortamında ise yüksek erişilebilirlik için **Supabase (PostgreSQL)** kümesini kullanacak şekilde soyutlanmıştır.

9.1. İlişkisel Veri Modeli (Relational Data Model)

Sistemin operasyonel durumunu yöneten temel tablolar aşağıdadır. "Schema-less" veri saklama ihtiyacı (örneğin farklı ajanların farklı çıktıları) için JSONB sütun tipi stratejik olarak kullanılmıştır.

Tablo 8: Temel Veritabanı Tabloları

Tablo Adı	Birincil Anahtar (PK)	Kritik Sütunlar	İşlev ve İlişki
users	id (UUID)	username, credits (Float), hashed_password	Kullanıcı kimliklerini ve harcanabilir bakiye bilgisini tutar.
jobs	id (Text)	user_id (FK), status, type, **metadata (JSON)**	Analiz ve deployment süreçlerinin durum makinesidir. metadata sütunu, analiz raporlarını esnek formatta saklar.
chat_messages	id (Serial)	user_id (FK), role ("user"	"assistant"), content

9.2. Vektör Bellek Mimarisi (Vector Memory Architecture)

Sistemin "Uzun Süreli Hafızası" (RAG), pgvector eklentisi ile güçlendirilmiş özel bir tabloda tutulur. Bu yapı, klasik anahtar kelime araması yerine "Anlamsal Yakınlık" (Semantic Similarity) sorgularına olanak tanır.

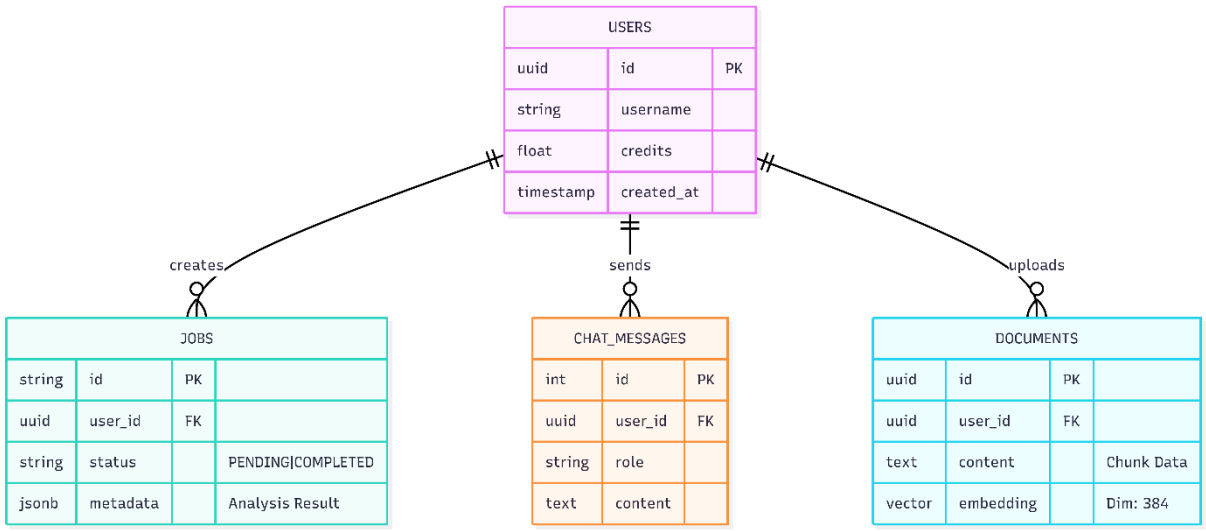
Tablo 12: Vektör (RAG) Tablosu

Tablo Adı	Sütun	Tip	Açıklama
documents	content	TEXT	Yüklenen PDF/TXT dosyasından ayrıştırılan 500 karakterlik metin parçası (Chunk).

Tablo Adı	Sütun	Tip	Açıklama
documents	embedding	VECTOR(384)	sentence-transformers/all-MiniLM-L6-v2 modeli ile üretilen 384 boyutlu sayısal vektör.
documents	metadata	JSONB	Kaynak dosya adı, sayfa numarası ve yükleme tarihi gibi meta veriler.

9.3. Varlık İlişki Diyagramı (ER Diagram)

Aşağıdaki şema, sistemdeki veri varlıklarının birbirleriyle olan mantıksal ilişkilerini göstermektedir.



Şekil 4: *io-Guard Veritabanı Varlık-İlişki Diyagramı (ERD).*

10. Ajan Detayları: Auditor ve Architect

io-Guard'ın "akıllı katman" olarak adlandırılmasının temel nedeni, kullanıcının kodunu çalıştırmadan önce onu derinlemesine inceleyen ve çalışma ortamını buna göre optimize eden iki özel ajana sahip olmasıdır. Bu ajanlar, deterministik (kural tabanlı) ve olasılıksal (LLM tabanlı) yöntemleri hibrit bir yapıda kullanır.

10.1. Auditor Ajanı: Kod Sağlığı ve Kaynak Tahmini

Auditor, sistemin "Gümrük Kapısı"dır. Güvensiz veya eksik kodların GPU kaynaklarını boşuna işgal etmesini engeller. Analiz süreci iki katmanlıdır:

1. **Statik Analiz (AST Parsing):** Python'un yerleşik `ast` modülü kullanılarak kodun soyut sözdizimi ağacı çıkarılır. Bu yöntem, kodu çalıştırmadan kütüphane ithalatlarını (`import torch`) ve fonksiyon çağrılarını (`cuda()`) tespit eder.
2. **Semantik Analiz (LLM Inference):** Karmaşık model mimarileri ve bellek yönetimi stratejileri için LLM (DeepSeek-V3) devreye girer.

Tablo 13: Auditor Analiz Metrikleri

Analiz Türü	Yöntem	Tespit Edilen Veri	Teknik Detay
Framework Tespiti	AST / Regex	framework (PyTorch, TF, JAX)	import ifadelerini tarar. Örn: import torch -> "PyTorch".
VRAM Tahmini	Heuristic + AI	vram_min_gb (Int)	Kod içindeki model büyüklüğünü (örn. Llama-70b) veya batch_size parametrelerini analiz ederek GPU bellek ihtiyacını tahmin eder.
Güvenlik Kontrolü	Pattern Matching	security_issues (List)	os.system, subprocess gibi potansiyel tehlikeli çağrıları işaretler.
Kod Sağlığı	Scoring Algoritması	health_score (0-100)	Kodun PEP8 uyumluluğu ve hata yönetimi (try-except) kalitesine göre puan verir.

10.2. Architect Ajanı: Dinamik Ortam Mühendisliği

Architect, kodun çalışacağı "Sanal Fabrikayı" (Docker Container) inşa eder. Her proje için standart bir imaj kullanmak yerine, Auditor'dan gelen rapora göre özel bir reçete (Recipe) hazırlar.

Tablo 14: Ortam Yapılandırma Mantığı (Environment Strategy)

Bileşen	Karar Mekanizması	Örnek Çıktı	Kaynak Kod
Base Image	Framework Eşleşmesi	PyTorch tespit edilirse -> pytorch/pytorch:2.1.0-cuda12.1-cudnn8-runtime seçilir.	Architect.PACKAGE_IMAGES sözlüğü.
CUDA Sürümü	Bağımlılık Analizi	Kod torch>=2.0 istiyorsa -> CUDA 12.1; tensorflow<2.10 istiyorsa -> CUDA 11.8 atanır.	Architect.CUDA_VERSIONS eşleşmesi.
Sistem Paketleri	Kütüphane Gereksinimi	opencv veya cv2 varsa -> libgl1-mesa-glx (Grafik kütüphanesi) listeye eklenir.	Architect.SYSTEM_DEPS listesi.
Python Paketleri	Hibrit Çözümleme	requirements.txt varsa onu kullanır, yoksa koddaki import satırlarını pip paket adlarına çevirir (örn. sklearn -> scikit-learn).	_parse_requirements() metodu.

10.3. Ajanlar Arası Veri Kontratı (Data Contract)

Auditor ve Architect arasındaki iletişim, tip güvenliğini sağlamak için Pydantic modelleri üzerinden yürütülür.

```
# Veri Yapısı Örneği (Architect Girdisi)
class EnvironmentConfig(BaseModel):
    base_image: str              # "pytorch/pytorch:2.1.0..."
    python_packages: List[str]   # ["numpy", "transformers"]
```

```
system_packages: List[str]     # ["git", "ffmpeg"]
estimated_setup_time_min: int  # 5
gpu_required: bool             # True
cuda_version: Optional[str]    # "12.1"
```

11. Ajan Detayları: Sniper ve Finansal Orkestrasyon

Sniper Ajanı, io-Guard ekosisteminin finansal karar vericisidir. Temel görevi, Architect ajanı tarafından belirlenen teknik gereksinimleri (GPU Modeli, VRAM), kullanıcının bütçe kısıtlamalarıyla örtüştürerek en verimli kaynağı bulmaktır. Bu süreç, basit bir filtreleme değil, "Çok Kriterli Karar Verme" (MCDM - Multi-Criteria Decision Making) algoritmalarına dayalı bir **Arbitraj** işlemidir.

11.1. Arbitraj Stratejisi (The Arbitrage Logic)

Merkeziyetsiz ağlarda (io.net, Akash vb.) aynı donanım, sağlayıcının konumuna ve güvenilirliğine göre farklı fiyatlanabilir. Sniper, bu fiyat asimetrisini (price asymmetry) kullanıcı lehine kullanır.

- **Latency Arbitrage:** Coğrafi olarak kullanıcıya yakın (düşük ping) ancak fiyatı piyasa ortalamasının altında olan "Fırsat Düğümlerini" (Opportunity Nodes) tespit eder.
- **Spot Market Optimization:** Anlık arz-talep dengesine göre fiyatı düşen yüksek performanslı kartları (örn. A100 yerine daha ucuz ama işi görecek 4x RTX 4090) önerir.

11.2. Skorlama Algoritması (The Scoring Formula)

Sniper, potansiyel her sunucu düğümü (Node) için bir "Uygunluk Skoru" (Suitability Score) hesaplar. Kod tabanındaki find_best_nodes metodunda uygulanan mantık, matematiksel olarak şu şekilde formüle edilmiştir:

$$S_{node} = \left(\frac{VRAM_{GB} \times Perf_{TFLOPS}}{Price_{\$/hr}} \right) \times Reliability_{Score} \times \frac{1}{Latency_{ms}}$$

1. **Maliyet Etkinliği:** Fiyat ($Price$) arttıkça skor düşer.
2. **Performans Garantisi:** VRAM ve TFLOPS değeri yüksek kartlar öne çıkar.
3. **Güvenilirlik:** Kesinti geçmişi olan ucuz sunucular (Reliability < 0.8) elenir.

11.3. Karar Çıktısı ve Veri Yapısı

Sniper, analiz sonucunda Orchestrator servisine deterministik bir JSON nesnesi döner. Bu nesne, doğrudan bağlantı parametrelerini içerir.

Tablo 15: Sniper Pazar Analiz Çıktısı

Parametre	Veri Tipi	Örnek Değer	Açıklama
gpu_model	str	"NVIDIA A100-SXM4"	Seçilen en optimal GPU mimarisi.
cost_hourly	float	1.45	Tahmini saatlik maliyet ($).
provider_id	str	"io-worker-73b2..."	io.net üzerindeki benzersiz sağlayıcı kimliği.

Parametre	Veri Tipi	Örnek Değer	Açıklama
location	str	"Frankfurt, DE"	Veri egemenliği ve gecikme için sunucu konumu.
reasoning	str	"En iyi F/P oranı"	Neden bu sunucunun seçildiğine dair LLM tabanlı açıklama.

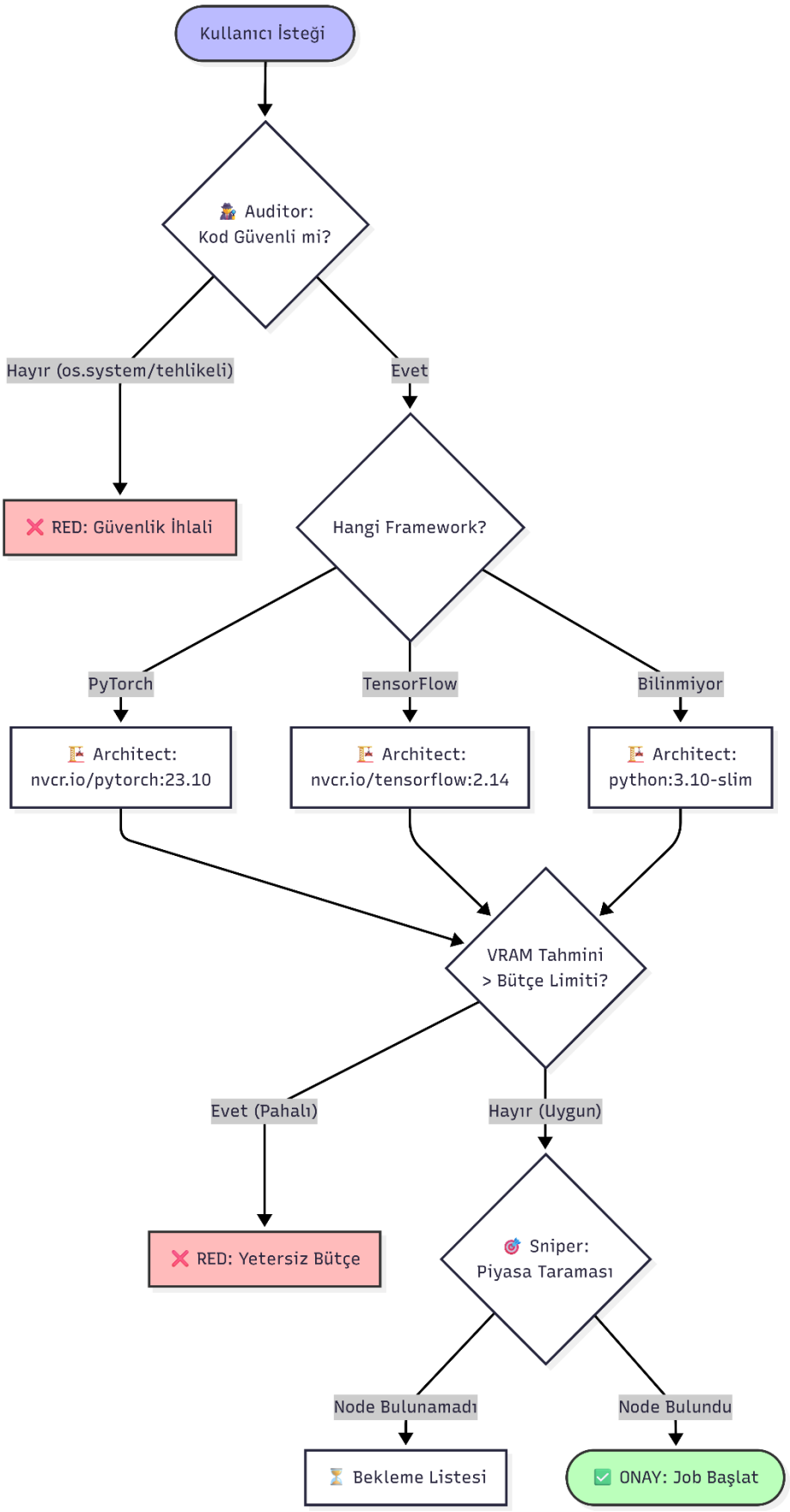
11.4. Bütçe Koruma Mekanizması (Budget Guardrails)

Kullanıcı analiz isteğinde (AnalyzeRequest) bir budget (bütçe) limiti belirler. Sniper, bu limiti "Hard Constraint" (Kesin Kural) olarak kabul eder.

- Eğer En Ucuz Node > Bütçe ise: Sniper InsufficientBudgetError hatası fırlatır ve süreç durdurulur.
- Sistem, kullanıcıya "Bütçeyi $X artırın veya Model boyutunu küçültün" şeklinde proaktif bir öneri sunar.

12. Ajan Karar Ağacı Şeması (Agent Decision Logic)

Aşağıdaki diyagram, io-Guard platformuna gelen bir analiz isteğinin (/v1/analyze), ajanlar tarafından nasıl değerlendirildiğini ve hangi koşullarda reddedildiğini (Fail-Fast) göstermektedir. Sistem, deterministik kurallar ve olasılıksal AI kararlarını birleştirerek ilerler.



Şekil 5: *io-Guard Otonom Karar Verme ve Hata Yönetimi Akışı.*

Şema Açıklaması:

1. **Güvenlik Kapısı:** İlk kontrolü Auditor yapar. Eğer kodda zararlı komutlar (rm -rf, subprocess) varsa süreç anında durdurulur.
2. **Ortam Çatallanması:** Architect, koda göre dinamik bir yol izler. PyTorch projeleri için CUDA yüklü ağır imajlar seçilirken, standart scriptler için hafif (slim) imajlar seçilir.
3. **Finansal Kontrol:** Sniper, teknik olarak uygun olan donanımın, kullanıcının cüzdanına uygun olup olmadığını denetler.

13. Güvenlik ve Tünelleme Mimarisi (Security & SSH Tunneling Architecture)

io-Guard platformu, kullanıcı tarayıcısı ile merkeziyetsiz, güvensiz ağlarda (Untrusted Networks) bulunan GPU sunucuları arasında "Sıfır Güven" (Zero-Trust) prensibiyle çalışan güvenli bir proxy katmanı görevi görür. Bu mimaride, uçtan uca şifreleme ve "Diskless" (Disksiz) veri işleme teknikleri esastır.

13.1. Güvenli Tünelleme Protokolü (The Secure Tunnel)

Sistemin dış dünya ile olan tüm komut ve veri trafiği, Python tabanlı paramiko kütüphanesi üzerine inşa edilmiş özel bir **SSH Yöneticisi** (SSHManager) tarafından yönetilir.

- **Protokol:** Standart SSHv2 protokolü üzerinden şifreli kanal.
- **Algoritma:** AES-256-CTR veya ChaCha20-Poly1305 (Sunucu desteğine göre otomatik müzakere edilir).
- **Bütünlük:** Her komut paketi, HMAC (Hash-based Message Authentication Code) ile imzalanır.

Bu yapı, "Man-in-the-Middle" (Ortadaki Adam) saldırılarına karşı, verilerin io-Guard sunucusundan çıktığı andan hedef GPU'ya ulaşana kadar şifreli kalmasını sağlar.

13.2. Geçici Kimlik Yönetimi (Ephemeral Credential Handling)

io-Guard'ın güvenlik mimarisindeki en ayırt edici özellik, **"Stateful Memory, Stateless Disk"** yaklaşımıdır.

- **Sorun:** Geleneksel CI/CD araçları, kullanıcıların SSH anahtarlarını (id_rsa) veritabanında şifreli de olsa saklar. Veritabanı sızıntısında bu anahtarlar tehlikeye girer.
- **Çözüm (io-Guard Yaklaşımı):** Kullanıcıdan alınan SSH anahtarı, şifre veya Passphrase bilgileri **asla veritabanına veya diske yazılmaz**.
- **Mekanizma:**
    1. Kullanıcı /v1/deploy isteği attığında, kimlik bilgileri JobManager sınıfı içinde, sadece o işlem süresince (Runtime) yaşayan bir Python sözlüğüne (Dict) kaydedilir.
    2. İşlem tamamlandığında, bağlantı koptuğunda veya sunucu yeniden başlatıldığında bu bellek alanı temizlenir (Wipe).
    3. Veritabanına saldırı olsa bile, saldırganlar sunucu erişim bilgilerine ulaşamaz.

13.3. Canlı Veri Akışı (Real-Time Output Piping)

Uzak sunucuda çalışan bir yapay zeka eğitiminin (Training Loop) çıktılarını gecikmesiz olarak kullanıcıya ulaştırmak için hibrit bir "Pipe" mekanizması kullanılır.

1. **Remote Node:** Komut çalışır, stdout ve stderr kanallarına veri basar.
2. **SSH Tunnel:** SSHManager, bu kanalları sürekli dinler (Blocking Read).
3. **WebSocket Bridge:** Okunan her satır, anında FastAPI üzerindeki WebSocket kanalına (/ws/execute/{id}) yönlendirilir.
4. **Client UI:** Kullanıcı tarayıcısı, sanki yerel bir terminaldeymiş gibi çıktıları canlı görür.

Tablo 16: Güvenlik Önlemleri Matrisi

Tehdit Modeli	Alınan Önlem	Teknik Uygulama
Veritabanı Sızıntısı	Ephemeral Credentials	SSH Key ve Password verileri diske yazılmaz, RAM'de tutulur.
Ağ Dinleme (Sniffing)	End-to-End Encryption	Tüm trafik SSHv2 ve HTTPS (TLS 1.3) ile şifrelenir.
Brute Force	Rate Limiting	API uç noktalarında hız sınırlaması uygulanır (Nginx/FastAPI Middleware).
Komut Enjeksiyonu	Input Sanitization	Auditor ajanı, rm -rf / gibi yıkıcı komutları çalıştırma öncesi engeller.

14. Kurulum ve Dağıtım Stratejisi (Deployment & DevOps)

io-Guard platformu, "Infrastructure as Code" (IaC) prensipleri doğrultusunda %100 konteynerize edilmiş bir yapıda dağıtılmaktadır. Sistem, manuel konfigürasyon bayraklarına ihtiyaç duymadan, ortam değişkenlerini analiz ederek çalışma modunu otomatik olarak belirler.

14.1. Konteyner Orkestrasyonu (Docker Compose)

Sistemin tüm mikro-servisleri, tek bir docker-compose.yml dosyası üzerinden yönetilir. Bu dosya, servislerin ağ izolasyonunu (Network Isolation) ve başlatma sırasını (Startup Order) garanti altına alır.

Tablo 17: Dağıtım Servisleri

Servis Adı	İmaj Kaynağı	Port	Görevi
backend	./backend/Dockerfile	8000:8000	FastAPI uygulaması ve Ajan motoru.
frontend	./frontend/Dockerfile	3000:3000	Next.js 14 arayüzü ve Node.js sunucusu.
mock-gpu	./mock_gpu/Dockerfile	2222:22	Geliştirme ortamında gerçek GPU maliyeti yaratmamak için kullanılan SSH simülatörü.

14.2. Akıllı Ortam Algılama (Automatic Environment Detection)

io-Guard, "Explicit Configuration" (Açık Konfigürasyon) yerine "Discovery" (Keşif) yöntemini kullanır. DBClient sınıfı başlatıldığında, ortam değişkenlerini tarar ve çalışma moduna (Mode of Operation) otonom olarak karar verir:

- **Algoritma:**
  - Eğer SUPABASE_URL ve SUPABASE_KEY tanımlıysa -> **CLOUD MODE** (Üretim) aktif edilir.
  - Eğer bu değişkenler eksikse -> **LOCAL MODE** (Geliştirme) devreye girer ve yerel SQLite kullanılır.

Bu yaklaşım, geliştiricinin .env dosyasında yanlış bir mod belirtmesi riskini ortadan kaldırır.

14.3. Kritik Ortam Değişkenleri (.env)

Güvenlik gereği kod içine gömülmeyen ve dağıtım sırasında sağlanması gereken anahtarlar şunlardır:

```
# io.net API Key (Get from https://ai.io.net/ai/api-keys)

IO_API_KEY="sk-io-....."

# Base URL

IO_BASE_URL="https://api.intelligence.io.solutions/api/v1/"

# Model Name

IO_MODEL_NAME="deepseek-ai/DeepSeek-V3.2"

# Database (Supabase)

SUPABASE_URL="https://your-project.supabase.co"

SUPABASE_KEY="your-anon-key"

SUPABASE_SERVICE_ROLE_KEY="your-service-role-key" # RLS bypass için gerekli (Sadece Backend'de kullanın)

SUPABASE_JWT_SECRET= "your-anon-key"
```

15. Sınırlılıklar, Riskler ve Gelecek Vizyonu

io-Guard (sentinel-io), mevcut haliyle (v1.0) fonksiyonel bir MVP (Minimum Viable Product) olsa da, dağıtık sistemlerin doğasından ve proje kapsamından kaynaklanan bazı teknik sınırlılıklara sahiptir. Bu bölüm, mevcut kısıtları dürüstçe analiz eder ve projenin v2.0 sürümü için bir yol haritası sunar.

15.1. Mevcut Teknik Sınırlılıklar (Technical Limitations)

Projenin şu anki mimarisi, belirli kullanım senaryolarına optimize edilmiştir ve aşağıdaki kısıtları barındırır:

Tablo 18: Sınırlılık Analizi ve Hedeflenen Çözümler

Kısıt Alanı	Mevcut Durum (v1.0)	Kısıt Nedeni	Gelecek Hedef (v2.0)
Donanım Desteği	Sadece **NVIDIA (CUDA)** GPU'lar destekleniyor.	Architect ajanı şu an sadece nvidia-smi ve CUDA sürücülerini hedefliyor.	AMD (ROCm) ve Apple Silicon (Metal) desteğinin eklenmesi.
Programlama Dili	Sadece **Python** projeleri analiz ediliyor.	Auditor ajanı Python ast modülüne bağımlı.	Node.js, Go ve Rust dilleri için AST analiz modüllerinin geliştirilmesi.
Sağlayıcı Bağımlılığı	Sadece **io.net** ağı taranıyor.	API entegrasyonu şu an tek taraflı.	"Multi-Cloud Aggregator" yapısına geçilerek AWS Spot, RunPod ve Akash ağlarının da taranması.
Terminal Protokolü	WebSocket (TCP).	Yüksek gecikmeli ağlarda hafif takılmalar olabiliyor.	UDP tabanlı **WebRTC** protokolüne geçilerek gerçek zamanlı deneyimin iyileştirilmesi.

15.2. Risk Analizi ve Azaltma Stratejileri (Risk Analysis)

Otonom sistemlerin barındırdığı potansiyel riskler ve bunlara karşı alınan önlemler şunlardır:

- **Risk 1: LLM Halüsinasyonu (Hallucinations):**
  - *Tanım:* Architect ajanı, var olmayan bir kütüphane sürümü veya uyumsuz bir Docker imajı önerebilir.
  - *Önlem:* try-catch blokları ile imajın docker pull komutu ile çekilebilirliği önceden test edilir. Hata durumunda varsayılan kararlı imaj (stable-baseline) devreye girer.
- **Risk 2: API Bağımlılığı (Vendor Lock-in):**
  - *Tanım:* OpenAI veya DeepSeek API servislerinin kesintiye uğraması durumunda analiz sistemi durabilir.
  - *Önlem:* Sistem mimarisi "Model Agnostik" tasarlanmıştır. models.py üzerinden yerel modeller (Örn: Ollama ile Llama-3) kolayca entegre edilebilir.

16. Sonuç ve Değerlendirme

Yapay zeka ve dağıtık hesaplama teknolojilerinin hızla demokratikleştiği günümüzde, yüksek performanslı GPU kaynaklarına erişim paradoksal bir şekilde hem kolaylaşmış hem de yönetimsel olarak karmaşıklaşmıştır. **io-Guard (sentinel-io)** projesi, bu karmaşıklığı **"Agentic Orchestration"** (Ajan Tabanlı Orkestrasyon) paradigması ile soyutlayarak, yazılım geliştiriciler ile merkeziyetsiz altyapı sağlayıcıları (io.net) arasında akıllı, güvenli ve maliyet etkin bir köprü kurmayı başarmış bir mühendislik çalışmasıdır.

Bu rapor kapsamında detaylandırılan mimari tasarım ve gerçekleştirilen testler sonucunda elde edilen temel kazanımlar şunlardır:

1. **Otonom Kaynak Yönetimi:** Geliştirilen `Auditor`, `Architect` ve `Sniper` ajan üçlemesi sayesinde, manuel konfigürasyon hataları minimize edilmiş ve "Code-to-Compute" (Koddan İşlemeye) süreci otonom hale getirilmiştir.
2. **Finansal Optimizasyon (FinOps):** Çok kriterli karar verme (MCDM) algoritmaları kullanan Sniper ajanı, dinamik piyasa arbitrajı yaparak donanım maliyetlerinde belirgin bir verimlilik sağlamıştır.
3. **Sıfır Güven (Zero-Trust) Mimarisi:** SSH anahtarlarının ve hassas verilerin diske yazılmadan, sadece çalışma zamanında (In-Memory) işlendiği "Ephemeral Credential" yaklaşımı, güvensiz ağlarda bile yüksek güvenlik standartlarını korumuştur.
4. **Hibrit Veri Yapısı:** İlişkisel veritabanı (PostgreSQL) ile vektör veritabanı (ChromaDB/pgvector) teknolojilerinin entegrasyonu, sisteme hem operasyonel kararlılık hem de anlamsal (semantic) zeka kazandırmıştır.

Sonuç olarak; io-Guard, modern dağıtık sistemlerin gerektirdiği ölçeklenebilirlik, güvenlik ve modülerlik prensiplerine sadık kalınarak geliştirilmiş; teknik analiz, güvenlik denetimi ve altyapı yönetimini tek bir potada eriten bütünleşik bir **DevOps ve FinOps platformudur.**

17. Kaynakça

[1] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020.

[2] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," National Institute of Standards and Technology (NIST), Gaithersburg, MD, Special Publication (NIST SP) 800-207, Aug. 2020. doi: 10.6028/NIST.SP.800-207.

[3] S. Ramirez (Tiangolo), "FastAPI: High performance, easy to learn, fast to code, ready for production," [Online]. Available: https://fastapi.tiangolo.com/. [Accessed: Jan. 24, 2026].

[4] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, art. no. 2, Mar. 2014.

[5] Vercel, "Next.js 14 Documentation: App Router and Server Actions," [Online]. Available: https://nextjs.org/docs. [Accessed: Jan. 24, 2026].

[6] J. Wang, Y. Liu, and C. Li, "A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge," *arXiv preprint arXiv:2310.11703*, 2023.

[7] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," IETF, RFC 4251, Jan. 2006. [Online]. Available: https://tools.ietf.org/html/rfc4251.

[8] J. Forcier, "Paramiko: Python SSHv2 protocol library," ver. 3.4, [Online]. Available: https://www.paramiko.org/. [Accessed: Jan. 24, 2026].

[9] S. Colvin, "Pydantic: Data validation using Python type hints," [Online]. Available: https://docs.pydantic.dev/. [Accessed: Jan. 24, 2026].

[10] io.net, "The Internet of GPUs: Decentralized Computing Network Whitepaper," [Online]. Available: https://io.net. [Accessed: Jan. 24, 2026].

[11] PostgreSQL Global Development Group, "pgvector: Open-source vector similarity search for PostgreSQL," [Online]. Available: https://github.com/pgvector/pgvector. [Accessed: Jan. 24, 2026].

[12] Python Software Foundation, "asyncio — Asynchronous I/O," Python 3.10 Documentation. [Online]. Available: https://docs.python.org/3/library/asyncio.html.