reNgine-ng: Technical Architecture

High-Level Architecture

System Architecture Diagram

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'16px'}}}%%
graph LR
    subgraph Client["Client Layer"]
        UI["Web Browser<br/>UI"]
        API["API Clients<br/>Automation"]
    end

    subgraph Gateway["Azure Gateway"]
        LB["Application Gateway<br/>SSL/TLS + WAF"]
    end

    subgraph App["Application Layer - VM (Docker)"]
        NGINX["Nginx<br/>Reverse Proxy"]
        WEB["Django Web<br/>Gunicorn+Uvicorn<br/>API+WebSocket"]
        WORKER["Celery Workers<br/>5-30 Concurrent"]
        BEAT["Celery Beat<br/>Scheduler"]
        OLLAMA["Ollama LLM<br/>GPU AI<br/>(Optional)"]
    end

    subgraph Data["Data Layer - Azure PaaS"]
        PG[("PostgreSQL<br/>Flexible Server")]
        REDIS[("Redis Cache<br/>Premium")]
        BLOB["Blob Storage<br/>Screenshots"]
    end

    subgraph Tools["Security Tools"]
        SCANTOOLS["Tools Layer<br/>Subfinder, Nuclei<br/>Nmap, FFUF"]
    end

    subgraph External["External APIs"]
        OPENAI["OpenAI<br/>GPT API"]
        NOTIFY["Notifications<br/>Slack/Discord"]
        NETLAS["Netlas<br/>OSINT"]
```

```
    end

    subgraph Monitor["Azure Security"]
        MON["Azure Monitor<br/>Insights"]
        KV["Key Vault<br/>Secrets"]
        DEF["Defender<br/>Security"]
    end

    UI --> LB
    API --> LB
    LB --> NGINX
    NGINX --> WEB
    WEB --> PG
    WEB --> REDIS
    WEB --> BLOB
    WEB --> WORKER
    WEB --> OLLAMA
    WORKER --> REDIS
    WORKER --> PG
    WORKER --> BLOB
    WORKER --> SCANTOOLS
    WORKER --> OLLAMA
    WORKER --> OPENAI
    WORKER --> NOTIFY
    WORKER --> NETLAS
    BEAT --> REDIS
    BEAT --> PG
    WEB --> KV
    WORKER --> KV
    WEB -.-> MON
    WORKER -.-> MON
    NGINX -.-> MON

    classDef azure fill:#0078D4,stroke:#003366,stroke-width:3px,color:#ff
    classDef app fill:#28a745,stroke:#1e7e34,stroke-width:3px,color:#fff,
    classDef data fill:#ffc107,stroke:#ff8c00,stroke-width:3px,color:#00(
    classDef external fill:#6c757d,stroke:#495057,stroke-width:3px,color:

    class LB,PG,REDIS,BLOB,MON,KV,DEF azure
    class WEB,WORKER,BEAT,NGINX,OLLAMA,SCANTOOLS app
    class OPENAI,NOTIFY,NETLAS external
```

Detailed Scan Execution Flow

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'15px', 'fontFam
sequenceDiagram
    autonumber
    participant U as User
    participant W as Web UI
    participant D as Database
    participant Q as Queue
    participant WK as Worker
    participant T as Tools
    participant AI as AI/LLM
    participant N as Notify
    participant S as Storage

    U->>W: Initiate Scan
    W->>D: Create Record
    W->>Q: Enqueue Task
    W-->>U: Scan ID + WebSocket

    Q->>WK: Dispatch Task
    WK->>D: Status: In Progress
    WK-->>W: WS: Started

    Note over WK,T: Phase 1: Subdomain Discovery
    WK->>T: Run Subfinder/CTFR
    T-->>WK: Subdomain List
    WK->>D: Store Subdomains
    WK-->>W: WS: 10 Found

    Note over WK,T: Phase 2: Port Scanning
    WK->>T: Run Nmap
    T-->>WK: Open Ports
    WK->>D: Store Ports
    WK-->>W: WS: 25 Discovered

    Note over WK,T: Phase 3: Crawling
    WK->>T: Run Gospider/Katana
    T-->>WK: URLs/Endpoints
    WK->>D: Store Endpoints
    WK-->>W: WS: 150 Found

    Note over WK,T: Phase 4: Vuln Scan
    WK->>T: Run Nuclei
    T-->>WK: Vulnerabilities
    WK->>D: Store CVE/CWE
```

```
    WK-->>W: WS: 8 Found

    Note over WK,T: Phase 5: Screenshots
    WK->>T: Run Eyewitness
    T-->>WK: PNG Files
    WK->>S: Upload to Blob
    S-->>WK: URLs
    WK->>D: Link Screenshots

    Note over WK,AI: Phase 6: AI Analysis
    WK->>AI: Analyze Surface
    AI-->>WK: AI Report
    WK->>D: Store Analysis

    WK->>D: Status: Complete
    WK->>N: Trigger Alert
    N-->>U: Summary

    WK-->>W: WS: Complete
    W-->>U: Show Results

    U->>W: Generate PDF
    W->>D: Query Findings
    W->>S: Fetch Media
    W-->>U: Download PDF
```

Data Model Architecture

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'14px'}}}%%
erDiagram
    ORGANIZATION ||--o{ DOMAIN : contains
    DOMAIN ||--o{ SUBDOMAIN : has
    SUBDOMAIN ||--o{ ENDPOINT : exposes
    SUBDOMAIN ||--o{ IP_ADDRESS : resolves_to
    SUBDOMAIN }o--|| SCREENSHOT : has
    ENDPOINT ||--o{ VULNERABILITY : has
    VULNERABILITY }o--o{ CVE : references
    VULNERABILITY }o--o{ CWE : categorized_by
    VULNERABILITY }o--o{ TAG : tagged_with
    SUBDOMAIN ||--o{ PORT : listens_on
    PORT }o--|| TECHNOLOGY : runs
    SCAN_HISTORY ||--o{ SUBDOMAIN : discovers
```

```
SCAN_HISTORY ||--o{ ENDPOINT : finds
SCAN_HISTORY ||--o{ VULNERABILITY : identifies
SCAN_HISTORY }o--|| ENGINE_TYPE : uses
PROJECT ||--o{ DOMAIN : manages
PROJECT ||--o{ USER : grants_access

ORGANIZATION {
    int id PK
    string name
    text description
    datetime created_at
}

DOMAIN {
    int id PK
    int org_id FK
    string name
    datetime created_at
}

SUBDOMAIN {
    int id PK
    int domain_id FK
    string name
    string http_url
    string status
    bool is_important
    jsonb tech
    datetime found_at
}

ENDPOINT {
    int id PK
    int subdomain_id FK
    string url
    string method
    string content_type
    int length
    jsonb patterns
    datetime found_at
}

VULNERABILITY {
    int id PK
```

```
        string name
        string severity
        text description
        text remediation
        text references
        bool open_status
        bool h1_report_id
        datetime found_at
    }

    SCAN_HISTORY {
        int id PK
        int domain_id FK
        int engine_id FK
        string scan_type
        string status
        datetime start_time
        datetime stop_time
        int subdomain_cnt
        int endpoint_cnt
        int vuln_cnt
    }

    PROJECT {
        int id PK
        string name
        text description
        string slug
    }
```

Technology Stack

Backend Framework & Runtime

Component	Technology	Version	Purpose
Web Framework	Django	3.2.25	Core application framework, ORM, admin interface
WSGI/ASGI Server	Gunicorn + Uvicorn Workers	23.0.0 + 0.30.6	**Modern ASGI architecture** - Production HTTP/WebSocket server with async capabilities

---

Component	Technology	Version	Purpose
ASGI Framework	Django Channels	3.0.5	Native async support and WebSocket handling
API Framework	Django REST Framework	3.14.0	RESTful API implementation
API Documentation	drf-yasg	1.21.7	OpenAPI/Swagger spec generation
Task Queue	Celery	5.4.0	Distributed task execution
Task Scheduler	Django Celery Beat	2.6.0	Periodic task scheduling (cron-like)
WebSocket	Django Channels	3.0.5	Real-time bidirectional communication
Async Workers	Gevent	24.2.1	Concurrent greenlet-based execution

Frontend Technologies

Component	Technology	Purpose
UI Framework	Bootstrap 4/5	Responsive design system
JavaScript	Vanilla JS + jQuery	DOM manipulation and AJAX
Data Tables	DataTables.js	Advanced table rendering with search/sort/filter
Charts	Chart.js	Dashboard visualizations and analytics
Real-Time	WebSocket API	Live scan progress updates
Markdown	Marked.js	Rich text rendering for notes

Database & Caching

Component	Technology	Version	Purpose
Primary Database	PostgreSQL	12+	Relational data storage for all entities
Message Broker	Redis	5.0.3+	Celery task queue backend
Cache Backend	Redis	5.0.3+	Session storage and query caching
ORM	Django ORM	3.2.25	Database abstraction and migrations

Reconnaissance Tool Stack

Category	Tools	Purpose
Subdomain Discovery	Subfinder, CTFR, Sublist3r, TLSX, OneForAll, Netlas, Amass	Passive and active subdomain enumeration
Port Scanning	Nmap, Naabu	Service discovery and banner grabbing
HTTP Crawling	Gospider, Hakrawler, Waybackurls, Katana, GAU	URL discovery and endpoint enumeration
Vulnerability Scanning	Nuclei (3000+ templates), Dalfox (XSS), CRLFuzz, S3Scanner	Automated vulnerability detection
Directory Fuzzing	FFUF	Content discovery via bruteforce
Screenshot	Eyewitness	Visual reconnaissance
WAF Detection	Wafw00f	Web application firewall identification
CMS Detection	CMSeek	Content management system fingerprinting
OSINT	theHarvester, LinkedInt, h8mail	Email/employee enumeration

AI/LLM Integration

Service	Models	Deployment	Purpose
OpenAI	GPT-3.5-turbo, GPT-4	Cloud API	Vulnerability analysis and reporting
Ollama	Mistral, Llama2, Neural Chat, CodeLlama	Self-hosted	On-premise AI inference (air-gapped)
GPU Support	NVIDIA CUDA, AMD ROCm	Local hardware	Accelerated model inference

Container & Orchestration

Component	Technology	Purpose
Containerization	Docker 20.10+	Application packaging
Orchestration	Docker Compose	Multi-container deployment
Base Images	python:3.10-alpine, postgres:12.19-alpine, redis:alpine, nginx:alpine	Minimal attack surface

---

Component	Technology	Purpose
Registry	GitHub Container Registry (ghcr.io)	Pre-built image distribution
Architectures	AMD64, ARM64	Multi-platform support

Security & Authentication

Component	Technology	Purpose
Authentication	Django Auth + Session	User login and session management
Authorization	Django Permissions + Custom RBAC	Role-based access (Admin/Auditor/Viewer)
Password Storage	PBKDF2 SHA256	Secure password hashing
SSL/TLS	Nginx with OpenSSL	HTTPS encryption
CORS	django-cors-headers	Cross-origin request handling

Key Azure Services (Recommended Deployment)

Compute

Azure Service	Purpose	Configuration
Azure Virtual Machines (Primary)	**Recommended production deployment**	**Standard_D8s_v3 (8 vCPU, 32 GB RAM)**
Azure VM - GPU-Enabled	**AI-powered report generation with Ollama**	**Standard_NC6s_v3 (6 vCPU, 112 GB RAM, 1x V100 GPU)**
Azure VM - Entry Level	Small-scale or development deployment	Standard_D4s_v3 (4 vCPU, 16 GB RAM)
Azure Container Instances (ACI)	*Optional:* Quick deployment for testing	4 vCPU, 16 GB RAM per container group
Azure Kubernetes Service (AKS)	*Optional:* Enterprise-scale orchestration	3-10 node cluster (Standard_D4s_v3)
Azure Container Registry (ACR)	Private image storage (optional)	Premium tier for geo-replication

Data Services

Azure Service	Purpose	Configuration
Azure Database for PostgreSQL	Managed database	Flexible Server, 4 vCore, 32 GB RAM, 128 GB SSD
Azure Cache for Redis	Managed Redis	Premium P1 (6 GB) with persistence
Azure Blob Storage	Object storage for screenshots/reports	Hot tier, LRS redundancy
Azure Files	Shared storage for tool configs	Premium tier for low latency

Networking & Security

Azure Service	Purpose	Configuration
Azure Application Gateway	Load balancer + WAF	WAF v2, SSL termination
Azure Virtual Network	Network isolation	Private subnet for backend services
Azure Key Vault	Secret management	Store DB passwords, API keys
Azure Private Link	Private connectivity	Secure DB and Redis access

Monitoring & Operations

Azure Service	Purpose	Configuration
Azure Monitor	Metrics and logging	Container Insights for AKS
Application Insights	APM and tracing	Django middleware integration
Log Analytics Workspace	Centralized logging	30-day retention
Azure Advisor	Cost and performance optimization	Enabled by default

---

Proposed Azure Marketplace Offer Type

Recommendation: Virtual Machine-Based Solution Application

Offer Structure: Azure VM Solution Template with Docker Compose

Primary Deployment Options:

1. **Option 1: Virtual Machine with Docker Compose - PRIMARY DEPLOYMENT SOLUTION (Recommended for All Deployments)**

- **Pros**:
  - Fastest time-to-value (10-minute deployment)
  - Familiar deployment model for most IT teams
  - Full control over Docker container orchestration
  - Easy migration from on-premise environments
  - Single VM simplifies networking and monitoring
  - Docker Compose provides reliable multi-container management
- **Minimum Requirements**:
  - **8GB RAM** or more (recommended: 16GB for production workloads)
  - **4 vCPU** or more (recommended: 8 vCPU for high concurrency)
  - **100GB SSD** for database and scan results storage
  - Ubuntu 22.04 LTS or similar Linux distribution
- **Recommended Azure VM SKUs**:
  - **Standard_D4s_v3** (4 vCPU, 16GB RAM) - Small to medium deployments
  - **Standard_D8s_v3** (8 vCPU, 32GB RAM) - Production deployments
  - **Standard_D16s_v3** (16 vCPU, 64GB RAM) - Enterprise/high-volume deployments
- **Deployment Method**: ARM template deploys VM with Docker and Docker Compose pre-configured
- **Ideal For**: Teams of 1-50 users, all Azure deployments from POC to production workloads

2. **Option 2: GPU-Enabled Virtual Machine - RECOMMENDED FOR AI-POWERED REPORTING**

- **GPU Acceleration for AI Reports**: Enhanced AI analysis and report generation with local LLM inference
- **Recommended Azure VM SKUs**:
  - **Standard_NC6s_v3** (6 vCPU, 112GB RAM, 1x V100 GPU) - AI report generation
  - **Standard_NC4as_T4_v3** (4 vCPU, 28GB RAM, 1x T4 GPU) - Cost-effective AI option
- **Use Cases**:
  - Organizations requiring AI-powered vulnerability analysis
  - Automated report generation with GPT-quality insights
  - Air-gapped environments needing local LLM inference (Ollama)
  - High-volume scanning with intelligent prioritization
- **Cost Benefit**: Local LLM vs. OpenAI API ($0.50/report local vs. $2-5/report cloud API)
- **Deployment Method**: Same ARM template with GPU-enabled VM SKU selection
- **Ideal For**: Security teams requiring advanced AI analysis, enterprises with strict data sovereignty requirements

---

Optional Advanced Deployment Options (For Special Use Cases)

3. **Option 3: Azure Container Instances (ACI) - For Development/Testing Only**

- **Pros**:

        - Pay-per-second billing
        - No VM management overhead
        - Automatic resource allocation
    - **Cons**:
        - Limited to 4 vCPU per container group
        - Less control over container orchestration
        - Higher costs for 24/7 workloads vs. VMs
    - **Ideal For**: Development environments, temporary testing, cost-sensitive POCs

4. **Option 4: Azure Kubernetes Service (AKS) - For Enterprise-Scale Multi-Tenant Deployments**

    - **Pros**:
        - Unlimited scalability
        - Advanced orchestration (auto-scaling, self-healing)
        - Multi-region deployment
        - GPU node pools for AI workloads
    - **Cons**:
        - Requires Kubernetes knowledge
        - Higher operational complexity
        - More expensive for small-scale deployments
        - Overkill for most use cases
    - **Ideal For**: Only for large MSPs/MSSPs with 100+ concurrent clients, organizations already standardized on Kubernetes, or multi-region HA requirements
    - **Note**: Most organizations should use VM deployment for simplicity and cost-effectiveness

Recommended Marketplace Listing: Solution Template + Managed Application

Why This Approach:

- **Solution Template**: Provides ARM template for one-click deployment of:
    - Azure Container Instances or AKS cluster
    - Azure Database for PostgreSQL
    - Azure Cache for Redis
    - Azure Blob Storage
    - Azure Key Vault
    - Azure Application Gateway
    - Pre-configured networking and security
- **Managed Application**: Optional managed service tier where:
    - Customer deploys to their Azure subscription (data sovereignty)
    - Vendor (Security Tools Alliance) can remotely manage updates and configuration
    - Customer retains full ownership of data and infrastructure

- Enables "Support + Managed Services" revenue stream

Monetization Strategy

Tier	Model	Price	What's Included
Community	BYOL (Bring Your Own License)	$0	Self-service deployment, community support
Professional	Annual Subscription	$5,000/year	Priority support, deployment assistance, quarterly updates
Enterprise	Custom Pricing	$25,000+/year	Managed service, SLA, custom features, compliance packs

Azure Consumption Revenue: Microsoft shares Azure infrastructure spend (20-25% margin)

Deployment Architecture on Azure

Recommended Production Architecture (VM-Based with Optional GPU)

```
%%{init: {'theme':'base', 'themeVariables': { 'fontSize':'15px'}}}%%
graph LR
    subgraph Public["Public Network"]
        INET["Internet<br/>Users"]
    end

    subgraph Azure["Azure Subscription: Customer-Owned"]
        subgraph RG["Resource Group: rengine-prod"]

            subgraph Net["Networking"]
                GW["App Gateway<br/>WAF v2 SSL<br/>(Optional)"]
                VN["VNet<br/>10.0.0.0/16"]
            end

            subgraph VM["Primary VM - D8s_v3"]
                subgraph Docker["Docker Compose Stack"]
                    NGX["Nginx<br/>Proxy"]
                    WEB["Django Web<br/>Gunicorn+Uvicorn"]
                    CEL["Celery Workers<br/>5-30 Parallel"]
                    BT["Celery Beat<br/>Scheduler"]
                    LPG[("Local PG<br/>(Optional)")]
                    LRD[("Local Redis<br/>Cache/Queue")]
```

```
            end
        end

        subgraph GPU["GPU VM - NC6s_v3 (Optional)"]
            OLL["Ollama LLM<br/>V100 GPU<br/>AI Reports"]
        end

        subgraph PaaS["Azure PaaS (Optional)"]
            PG[("PostgreSQL<br/>Flexible")]
            RD[("Redis<br/>Premium")]
            BL["Blob Storage<br/>Media"]
            KV["Key Vault<br/>Secrets"]
        end

        subgraph Mon["Monitoring"]
            MN["Monitor<br/>Insights"]
        end
    end
end

INET --> GW
GW --> NGX
INET -.->|Direct| NGX
NGX --> WEB
WEB --> LPG
WEB --> LRD
WEB -.->|Opt| PG
WEB -.->|Opt| RD
WEB -.->|Opt| BL
WEB -.->|Opt| KV
CEL --> LPG
CEL --> LRD
CEL -.->|Opt| PG
CEL -.->|Opt| RD
CEL -.->|Opt| BL
CEL -.->|Opt| OLL
BT --> LRD
BT -.->|Opt| RD
WEB -.-> MN
CEL -.-> MN

classDef azure fill:#0078D4,stroke:#003366,stroke-width:3px,color:#ff
classDef vm fill:#28a745,stroke:#1e7e34,stroke-width:3px,color:#fff,f
classDef docker fill:#2496ED,stroke:#1a73b8,stroke-width:3px,color:#f
```

```
    classDef gpu fill:#76B900,stroke:#5a8c00,stroke-width:3px,color:#fff,

    class GW,VN,PG,RD,BL,KV,MN azure
    class NGX,WEB,CEL,BT,LPG,LRD docker
    class OLL gpu
```

Scalability & Performance Characteristics

Concurrency Configuration

Component	Min	Max	Recommended	Notes
Celery Workers	5	30	15	Configurable via MIN_CONCURRENCY/MAX_CONCURRENCY env vars
Web Server (Gunicorn)	4	32	8	Workers = (2 × CPU cores) + 1
Database Connections	20	500	100	PostgreSQL max_connections
Redis Connections	50	1000	200	Redis maxclients

Estimated Performance Metrics

Workload	Configuration	Targets/Day	Azure Cost (Est.)
Individual/Small Team	VM (D4s_v3: 4 vCPU, 16GB) + Local DB/Redis	50 domains	~$140/month
Medium Team	VM (D8s_v3: 8 vCPU, 32GB) + Local DB/Redis	200 domains	~$280/month
Large Team with AI	VM (D8s_v3) + GPU VM (NC6s_v3) + Optional PaaS	500+ domains	~$1,200/month
Enterprise	VM (D16s_v3: 16 vCPU, 64GB) + GPU + Managed PaaS	1000+ domains	~$1,800/month

Cost Comparison vs. Container Orchestration:

- VM deployment: **60-70% cost savings** vs. AKS for equivalent workloads
- GPU VM for AI: **$800/month** (NC6s_v3) vs. OpenAI API costs of **$500-1500/month** for similar volume
- Managed PaaS services are optional - local containers reduce costs further

Security Architecture

Defense-in-Depth Strategy

1. **Network Layer**
    - Azure NSGs for firewall rules
    - Private endpoints for PaaS services (no public internet exposure)
    - Application Gateway WAF for OWASP Top 10 protection
2. **Application Layer**
    - Django security middleware (CSRF, XSS, clickjacking protection)
    - Role-based access control (RBAC)
    - Project-based data isolation
3. **Data Layer**
    - Encryption at rest (Azure Storage Service Encryption)
    - Encryption in transit (TLS 1.2+)
    - Automated backups with point-in-time restore
4. **Secrets Management**
    - Azure Key Vault for all credentials
    - Managed Identity for passwordless authentication
    - No hardcoded secrets in containers
5. **Monitoring & Compliance**
    - Azure Defender for Containers
    - Security Center recommendations
    - Audit logging to Log Analytics

Migration Path from On-Premise to Azure

Phase 1: VM Deployment (1-2 hours)

- Deploy Docker Compose to Azure VM using ARM template
- Minimal configuration required (environment variables only)
- Instant production-ready deployment

Phase 2: GPU Integration (Optional - 1 day)

- Add GPU-enabled VM for AI-powered reporting
- Deploy Ollama service for local LLM inference
- Configure Django to use GPU VM for report generation

Phase 3: Managed PaaS Enhancement (Optional - 1 week)

- Optionally migrate PostgreSQL to Azure Database for PostgreSQL
- Optionally migrate Redis to Azure Cache for Redis
- Optionally move screenshots to Azure Blob Storage
- Update connection strings as needed
- **Note**: Local containers are sufficient for most deployments

Phase 4: Enterprise Features (Optional - Ongoing)

- Add Application Gateway + WAF for enhanced security
- Implement Azure Monitor alerts and dashboards
- Configure automated backups and DR
- Add Azure Key Vault for secrets management

Advanced Phase (Only for Large MSPs): Container Orchestration

- For organizations with 100+ concurrent clients, consider AKS
- Requires Kubernetes expertise and significantly higher costs
- Most organizations should remain on VM deployment

Integration Points for Azure Ecosystem

Current Capabilities

- **Azure Active Directory**: Can integrate via SAML/OIDC (requires custom development)
- **Azure DevOps**: CI/CD pipelines for automated deployment
- **Azure Monitor**: Custom metrics and logging (via Python SDK)
- **Azure Key Vault**: Secrets injection via init containers

Future Enhancements (Marketplace Roadmap)

- Native AAD SSO integration

- Azure Sentinel integration for SIEM correlation
- Azure Front Door for global deployment
- Azure Cost Management integration for usage tracking