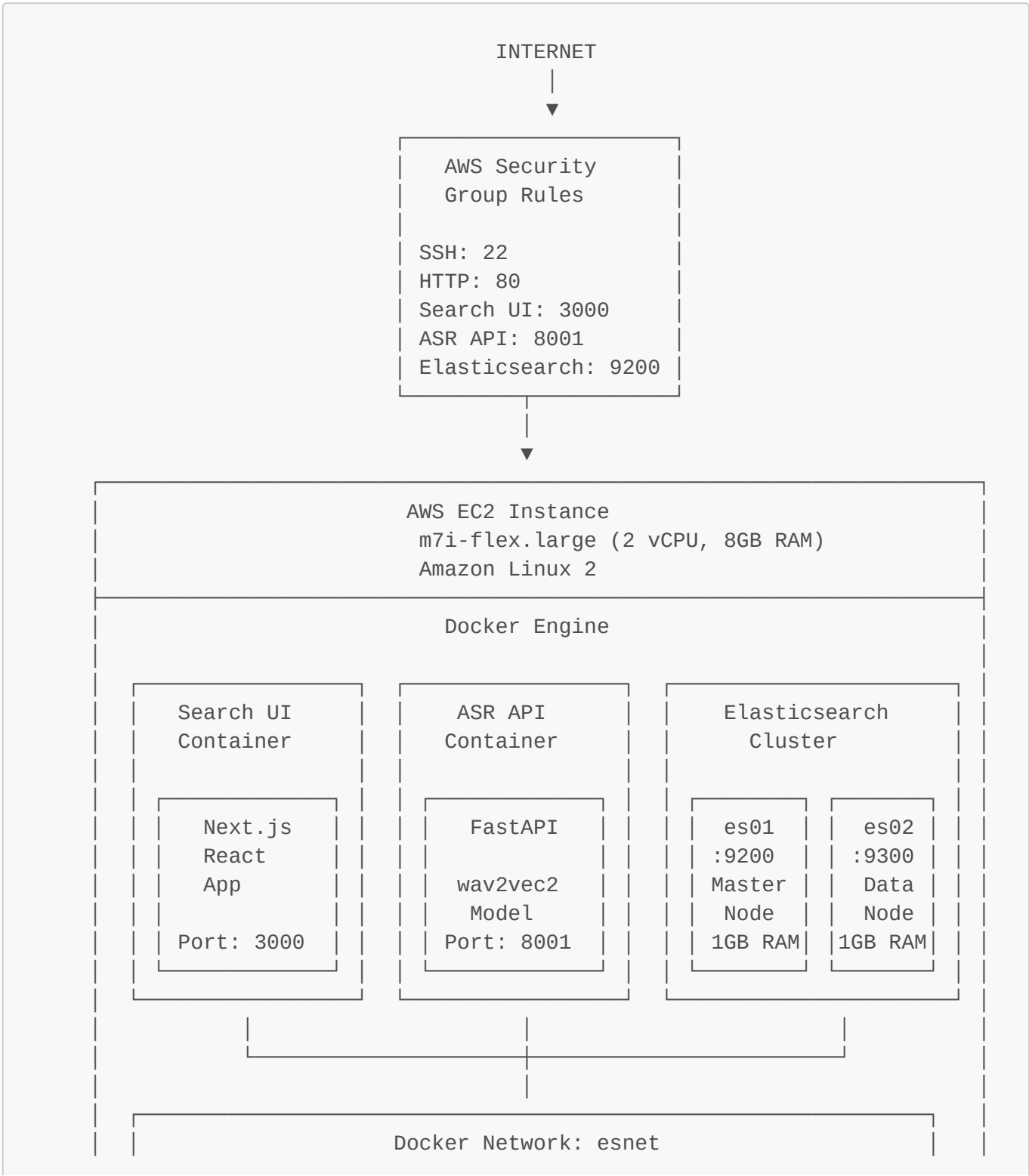
HTX Technical Test - AWS Deployment Architecture Design

Architecture Overview

This document presents the deployment architecture for the HTX Technical Test, featuring an Elasticsearch-based search system with a React frontend, deployed on AWS infrastructure without managed services.

Primary Architecture: Single VM Deployment (AWS Free Tier)



```
│   │                                                                   │   │
│   │  Service Discovery & Internal Communication                       │   │
│   │  - search-ui → elasticsearch:9200                                 │   │
│   │  - asr-api → elasticsearch:9200                                   │   │
│   │  - es01 ↔ es02 cluster communication                              │   │
│   └──────────────────────────────────────────────────────────────────┘   │
└──────────────────────────────────────────────────────────────────────────┘
```

Component Details

1. Search UI Frontend (Port 3000)

- **Technology**: Next.js 14 with React
- **Library**: @elastic/react-search-ui
- **Features**:
  - Full-text search on generated_text field
  - Faceted filtering on age, gender, accent, duration
  - Search-as-you-type with debouncing
  - Pagination and results per page controls
  - Responsive design with Tailwind CSS

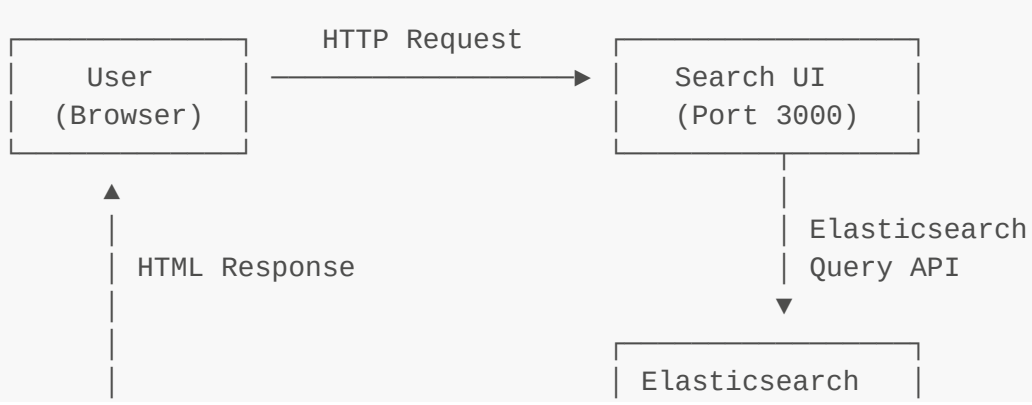
2. ASR API Backend (Port 8001)

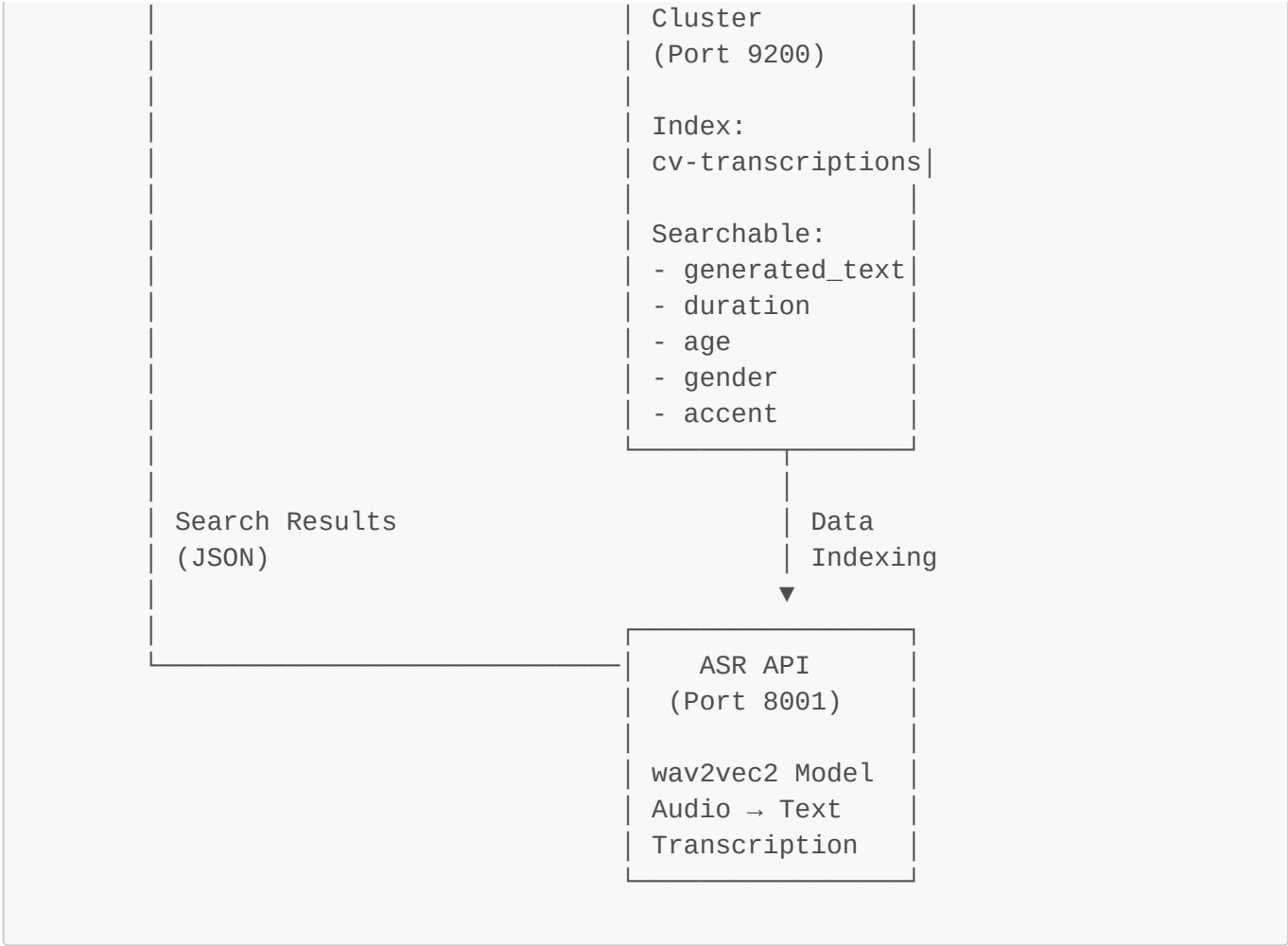
- **Technology**: FastAPI with wav2vec2-large-960h model
- **Endpoints**:
  - GET /ping - Health check
  - POST /asr - Audio transcription
- **Integration**: Processes Common Voice dataset

3. Elasticsearch Cluster (Port 9200)

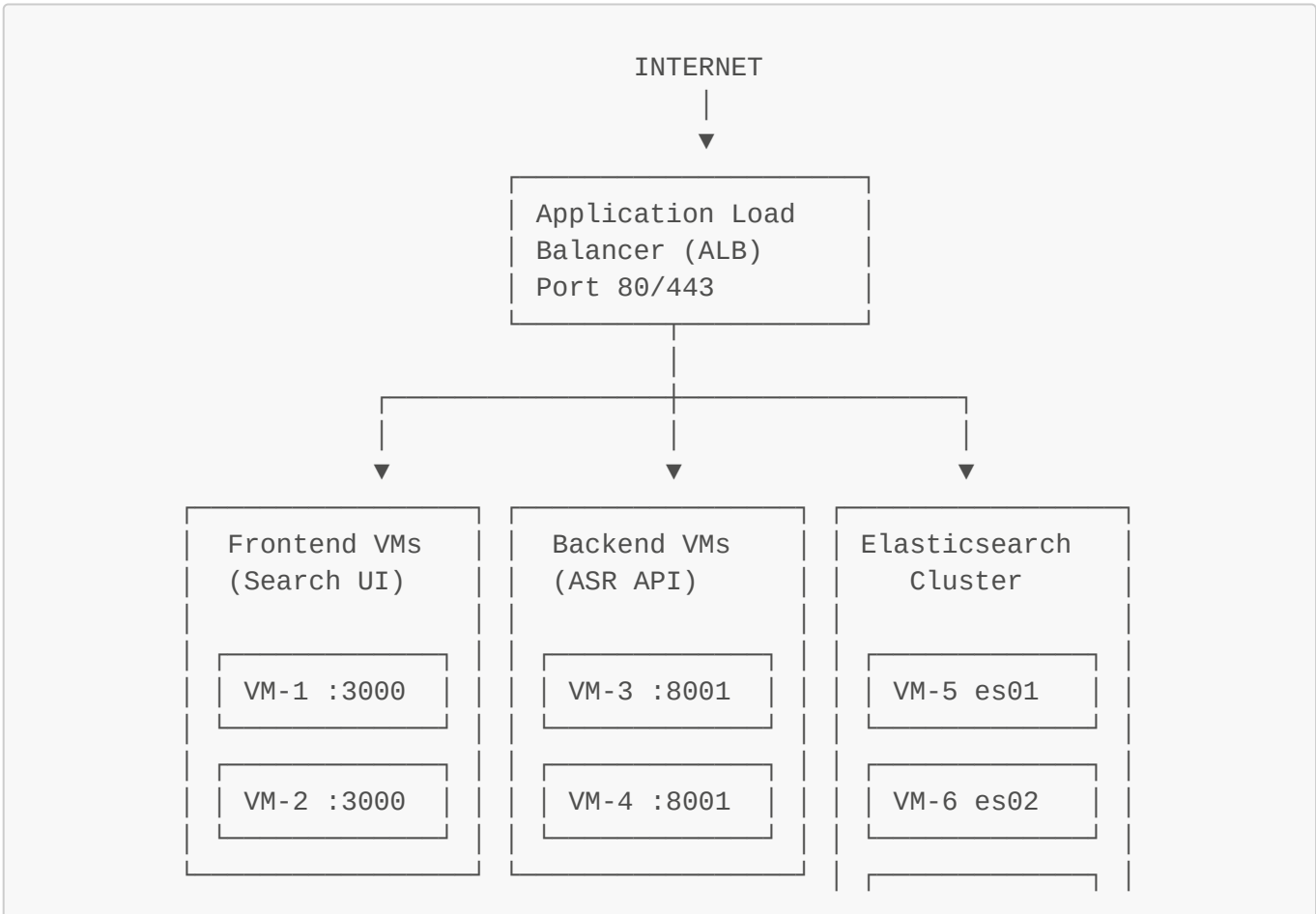
- **Configuration**: 2-node cluster (es01, es02)
- **Image**: docker.elastic.co/elasticsearch/elasticsearch:9.1.3
- **Index**: cv-transcriptions with 4,076 records
- **Memory**: Optimized 1GB heap per node for t2.micro
- **Fields**: generated_text, duration, age, gender, accent

Data Flow Architecture





Multi-VM Production Architecture (Scalable Option)



```
                                                    │ │ VM-7 es03   │ │
                                                    │ └─────────────┘ │
                                                    └─────────────────┘
```

Deployment Configuration

Docker Compose Services

```yaml
services:
  search-ui:
    build: ./search-ui
    ports: ["3000:3000"]
    depends_on: [es01]
    networks: [esnet]

  asr-api:
    build: ./asr
    ports: ["8001:8001"]
    volumes: ["./common_voice:/app/:ro"]

  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:9.1.3
    environment:
      - node.name=es01
      - cluster.name=cv-cluster
      - discovery.seed_hosts=es02
      - cluster.initial_master_nodes=es01,es02
      - ES_JAVA_OPTS=-Xms1g -Xmx1g
    ports: ["9200:9200"]
    networks: [esnet]

  es02:
    image: docker.elastic.co/elasticsearch/elasticsearch:9.1.3
    environment:
      - node.name=es02
      - cluster.name=cv-cluster
      - discovery.seed_hosts=es01
      - cluster.initial_master_nodes=es01,es02
      - ES_JAVA_OPTS=-Xms1g -Xmx1g
    networks: [esnet]
```

Security Configuration

AWS Security Group Rules

```
Inbound Rules:
- SSH (22): Your IP only
- HTTP (80): 0.0.0.0/0
- Custom (3000): 0.0.0.0/0  # Search UI
```

```
- Custom (8001): 0.0.0.0/0  # ASR API
- Custom (9200): 0.0.0.0/0  # Elasticsearch

Outbound Rules:
- All traffic: 0.0.0.0/0
```

Architecture Benefits

1. **Scalability**: Horizontal scaling across multiple VMs
2. **Fault Tolerance**: Service redundancy and health checks
3. **Cost Optimization**: Free tier compliance with production path
4. **Maintainability**: Clear separation of concerns
5. **Performance**: Dedicated resources per service tier
6. **Security**: Proper network isolation and access controls