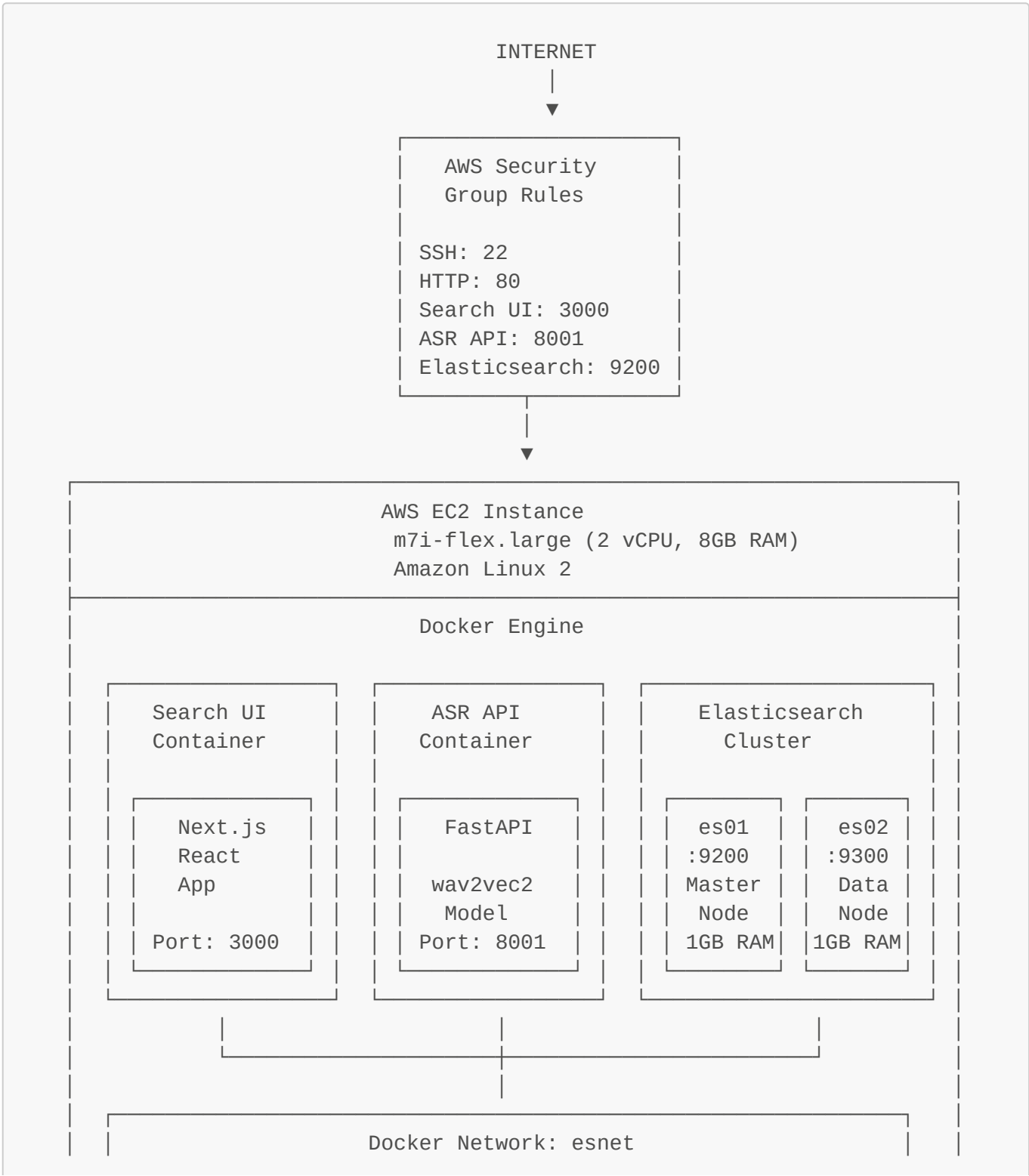
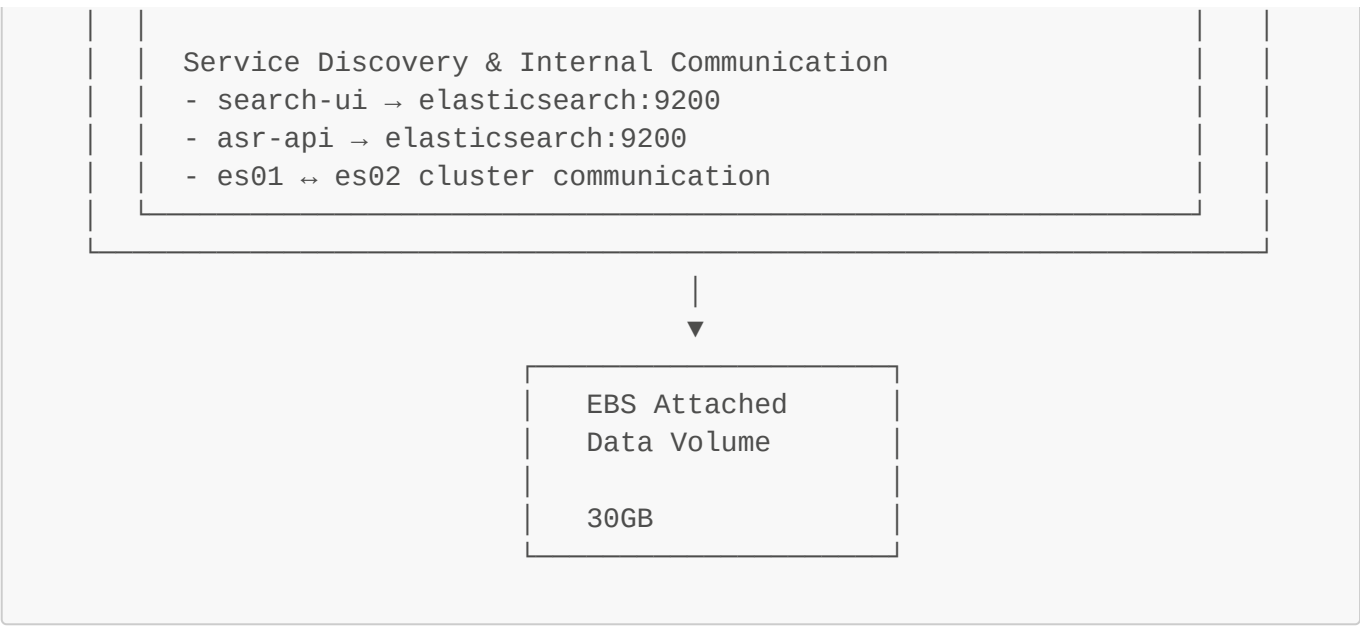
HTX Technical Test - AWS Deployment Architecture Design

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)





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

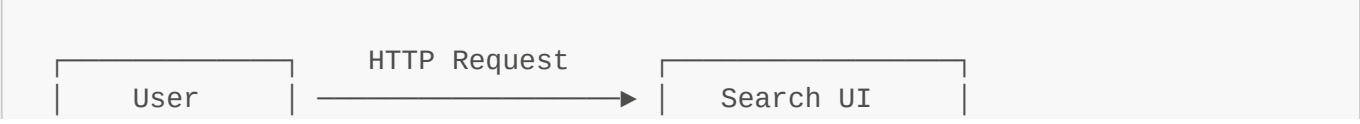
2. ASR API Backend (Port 8001)

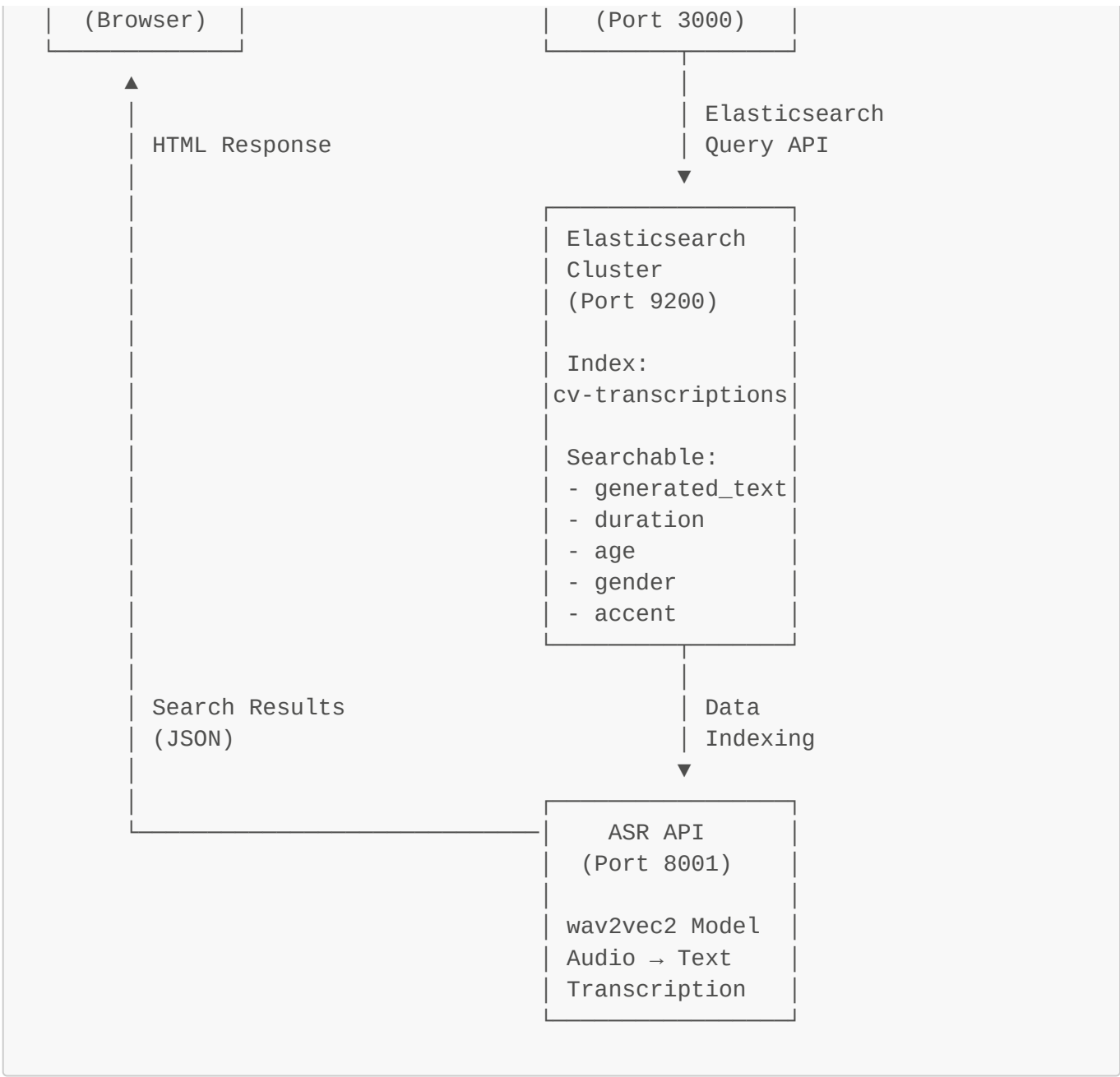
- **Technology**: FastAPI with wav2vec2-large-960h model
- **Endpoints**:
  - GET /ping - Health check
  - POST /asr - Audio transcription
- **Integration**: Processes Common Voice dataset

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 m7i-flex.large
- **Fields**: generated_text, duration, age, gender, accent

Data Flow Architecture





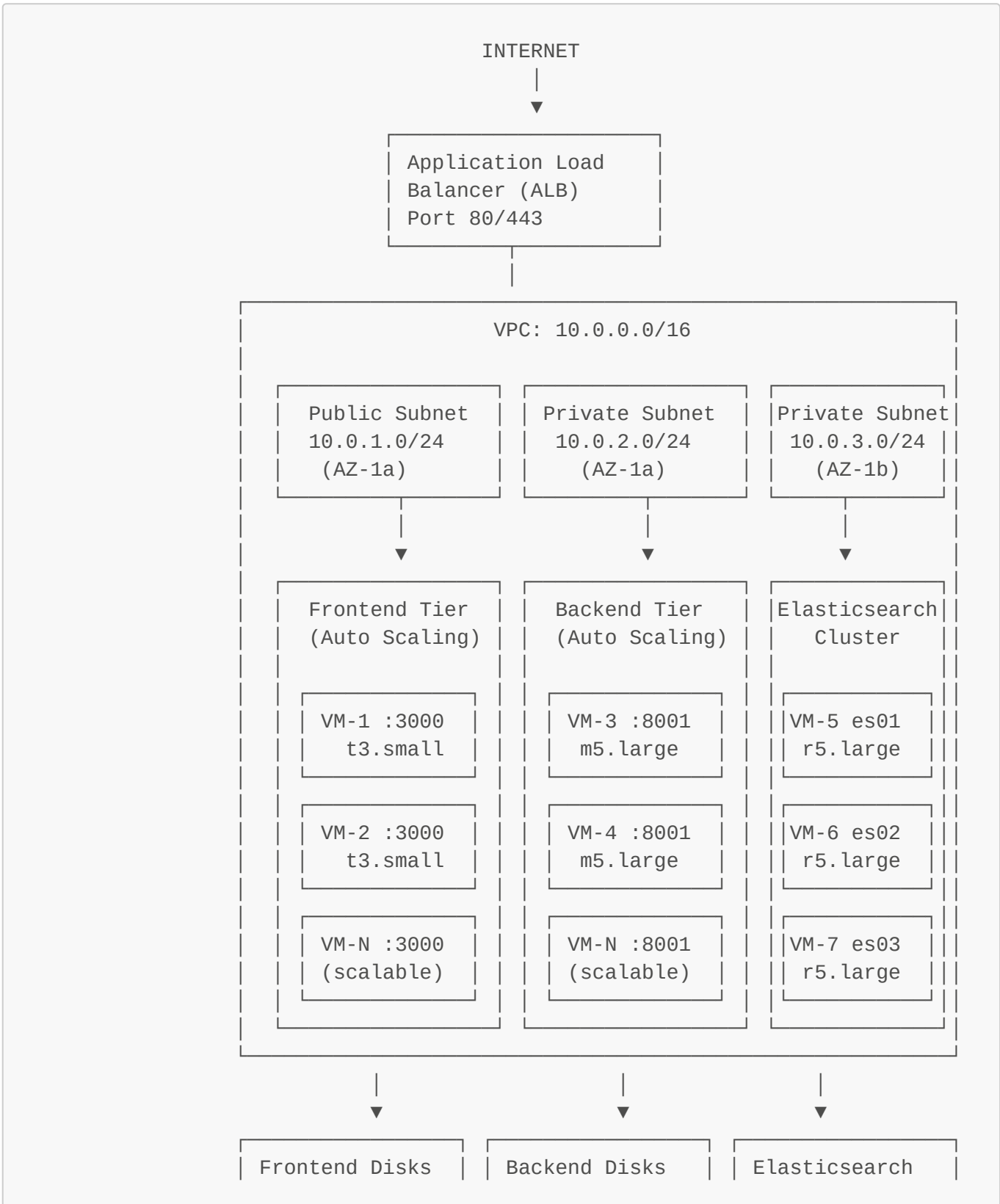
Single VM Deployment - Key Considerations and Assumptions

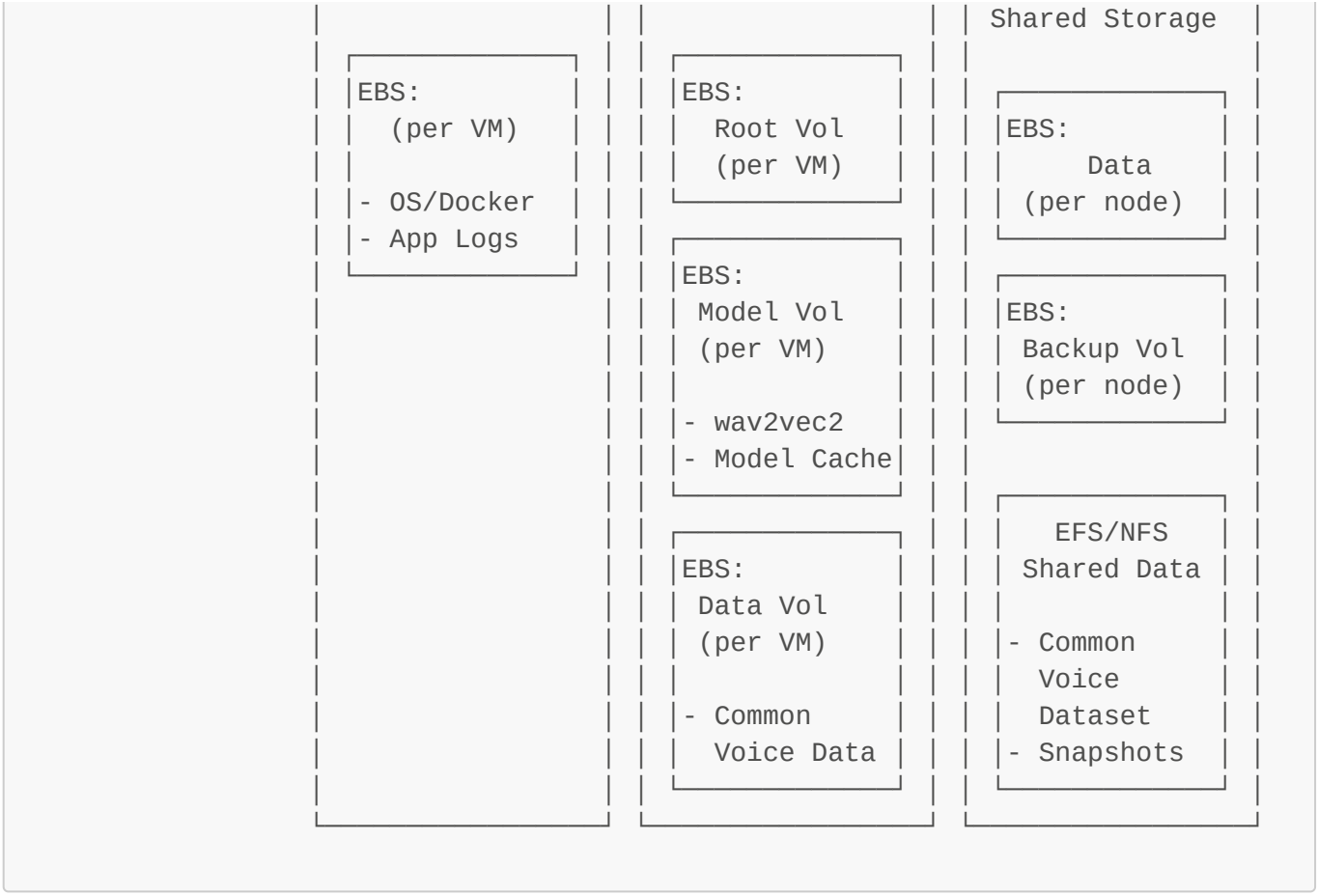
- There was no scalability requirement and so I opted for the simplest, most cost-effective solution for my deployment.
- Common Voice dataset consists of small, non-sensitive files and it was decided to stream from local compute to the EC2 endpoints as it would not incur storage costs.
- For sensitive data, it might be better to use HTTPS to encrypt data.
- Maximum free tier attached EBS storage was more than enough for the STT model, and other files.
- 8 GB of memory for the chosen EC2 instance was sufficient for running a 2-node Elasticsearch cluster and also the STT model transcription.
- Weakness: single point of failure with no redundancy.
- Weakness: all services compete for CPU/memory on the same host, leading to potential performance issues.
- Default VPC and subnet configuration was sufficient for a small deployment.

Single VM Deployment - Future Improvements

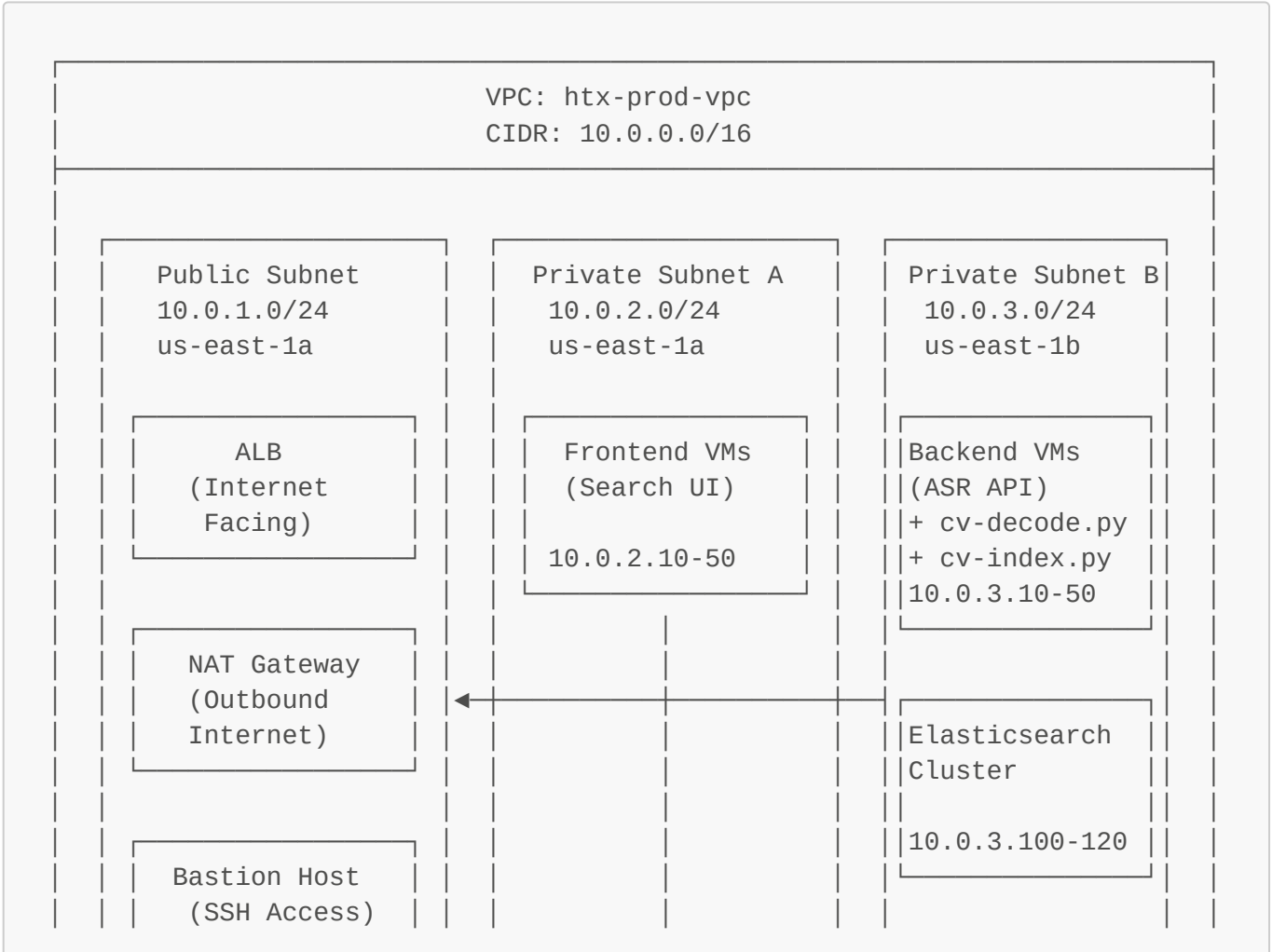
- Enable HTTPS for secure transmission.
- Enable monitoring via CloudWatch metrics for instance health.
- Work on horizontal scalability - see multi-VM approach below.
- Use a reverse proxy like Nginx to terminate TLS and to reduce attack surface (only exposes ports 22 and 443 and keeps 3000, 8001 and 9200 private)
- Use DNS for human-readable URL instead of having to type in the public IP.
- Further restrict inbound and outbound security group rules - right now, it is too permissive.

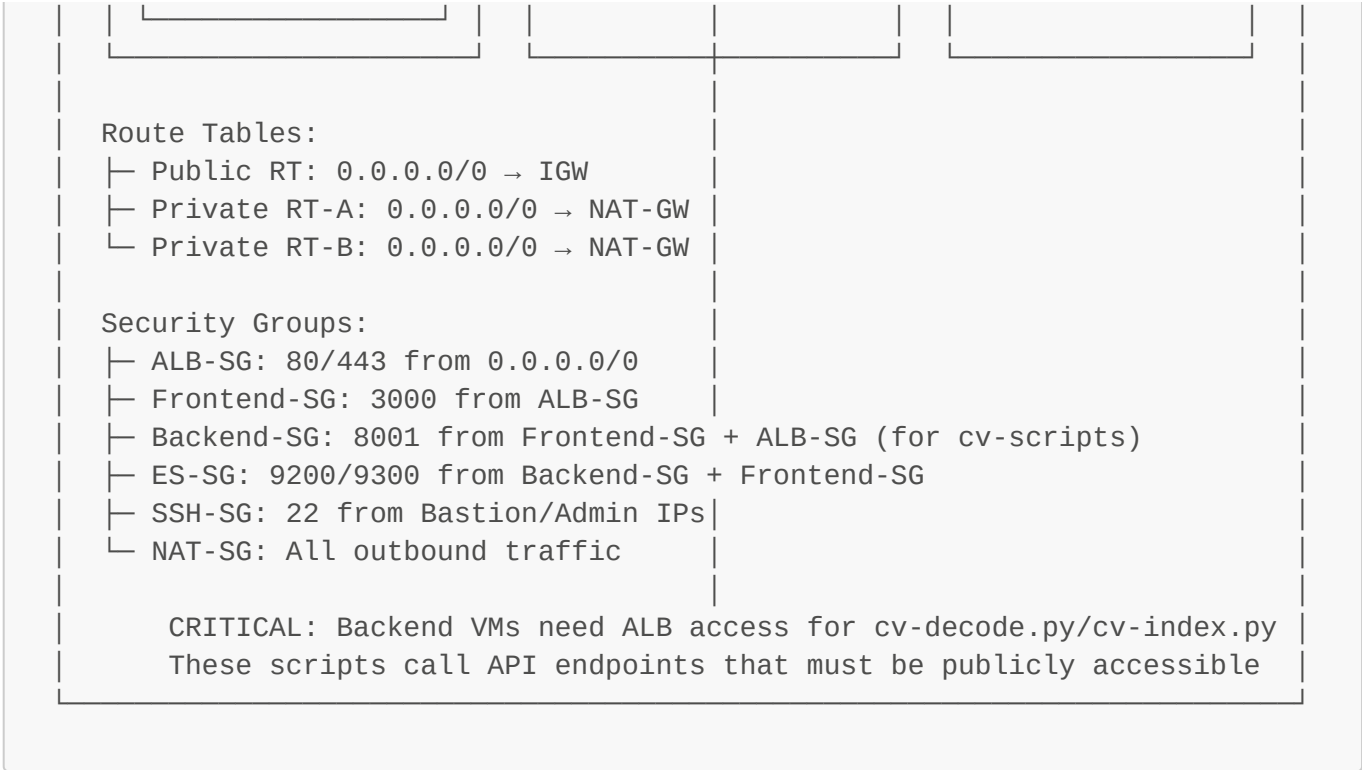
Multi-VM Production Architecture (Scalable Option)





Multi-VM Network Architecture Details





Multi-VM Production - Key Considerations and Assumptions

- Keep all the EC2 instances in private subnets and allow outward bound access via a NAT gateway only for instances that host the ASR_API service.
- Splitting up the microservices into separate EC2 instances helps to specialize the EC2 instances for their functions - for example, the ASR_API could make use of a GPU instance for faster transcription.
- Putting everything behind an Application Load Balancer helps to balance traffic and also reduces the attack surface as everything else is private within subnets.
- Bastion Host allows SSH access.
- For transcription, interactive (1 at a time) can be done from local compute to the ASR API service as there would be no additional storage cost and should be fast enough. Batch processing suggested to upload to S3 then call the transcription from there and then do cleanup as S3 is cheaper than EBS.
- Use gp3 EBS volumes for application and Elasticsearch data, tuning IOPS/throughput for each workload; this provides predictable performance at lower cost compared to gp2.
- Treat S3 as the system of record for large datasets (e.g., Common Voice) and Elasticsearch snapshots, while using EBS as local cache for active processing; this reduces costs and improves durability.
- Use EFS/NFS only for shared data distribution if strictly required, with local caching for performance.

Multi-VM Production - Future Improvements

- Use ECS for managed autoscaling and resource optimization.