**Health Tracker API – Submission Answers**

**1. Deployment on AWS**

To deploy the Health Tracker API on AWS efficiently, the following steps should be taken:

**Compute Service**

The API will be deployed using:

* **AWS Fargate (ECS)** – A serverless containerized service that handles scaling automatically.
* **Alternative:** Amazon EC2 if full control over the environment is required.

**Database Management**

* **Amazon RDS (MySQL/Aurora)** – A managed relational database with automatic backups and high availability.

**Load Balancing & API Management**

* **Application Load Balancer (ALB)** – Distributes API traffic across multiple instances.

**Storage & Logging**

* **Amazon CloudWatch** – Used for logging and performance monitoring.
* **Amazon S3** – Used for storing any files or large health data if required.
* **AWS X-Ray** – Provides API request tracing and latency analysis.

**Security & Access Control**

* **AWS IAM** – Ensures appropriate role-based access to resources.
* **AWS Secrets Manager** – Securely stores credentials.
* **AWS WAF** – Protects against web attacks such as DDoS and SQL injection.

**CI/CD for Automated Deployment**

* **AWS CodePipeline & CodeBuild** – Automates deployment processes from GitHub or another version control system.
* **Amazon Elastic Beanstalk** – If a simpler deployment is needed with automated scaling.

**2. Scaling & Troubleshooting**

As the application gains popularity, issues such as inaccurate health scores, delayed API responses, and system crashes may arise.

**Diagnosing the Issues**

**1. Health Scores Are Inaccurate**

* Verify the accuracy of stored data and ensure no missing fields.
* Improve the health score formula by grouping users based on demographics and health conditions.
* Ensure real-time data updates instead of relying on outdated averages.

**2. API Responses Are Delayed**

* Identify slow queries using **AWS X-Ray** and **CloudWatch Metrics**.
* Optimize SQL queries by using proper indexing.
* Implement **Redis caching** for frequently accessed data instead of querying the database repeatedly.

**3. Application Occasionally Crashes Under Load**

* Monitor CPU and memory usage via **AWS CloudWatch**.
* Implement **Auto Scaling Groups** to add or remove instances based on demand.
* Introduce **Rate Limiting** to prevent excessive requests from overwhelming the system.

**Long-Term Solutions for Scalability**

✅ **Health Score Optimization**

* Use **Batch Processing** to precompute and store health scores periodically instead of calculating them in real-time.
* Store aggregated data in **ElasticSearch or Redis** for faster retrieval.

✅ **Improving API Response Time**

* **Implement Caching** – Store frequently accessed data in Redis to reduce database queries.
* **Optimize Database Queries** – Add indexing and avoid costly operations such as SELECT \*.
* **Use Database Sharding** – Split large user datasets across multiple database servers.

✅ **Ensuring System Stability**

* **Enable Auto Scaling** – Automatically adjust resources based on traffic spikes.
* **Set Up Rate Limiting** – Restrict excessive API requests from a single user or IP.
* **Deploy Read Replicas** – Reduce the load on the primary database by distributing read queries.
* **Enable AWS CloudWatch Alerts** – Detect and respond to system failures before they escalate.

These measures will ensure that the Health Tracker API remains **accurate, scalable, and highly available** under high user demand.