Securing and Hardening the Talentlink System

Talentlink

Focus: Non-Functional requirements

Alizada,Erfan E.

2025

**Introduction**

Enterprise systems are not only measured by what features they provide but also by how well they handle non-functional requirements such as security, availability, scalability, and maintainability. The Talentlink project, built with a Flutter frontend and a FastAPI backend, was deployed on Oracle Kubernetes Engine (OKE). This essay explains how the project addressed non-functional requirements and how specific design choices align with enterprise architecture principles.

**Security and Confidentiality**

Security is a core non-functional requirement. In Talentlink, several measures were taken:

* **Ingress with TLS**: All external traffic enters through the NGINX Ingress controller. TLS certificates from Let’s Encrypt ensure encrypted communication between clients and the system.
* **Internal-only backend**: The backend service is only available inside the Kubernetes cluster. This limits exposure to the internet and reduces the attack surface.
* **Network policies**: Communication between services follows the principle of least privilege, ensuring only required connections are allowed.

These measures ensure **confidentiality, integrity, and controlled access**, which are fundamental to enterprise-level security.

**Availability and Reliability**

High availability and reliability are non-functional goals that ensure the system remains usable under failure conditions:

* **Replica sets**: Both frontend and backend services are deployed with multiple replicas for redundancy.
* **Liveness and readiness probes**: Kubernetes continuously checks if services are functioning (liveness) and ready to accept traffic (readiness). If a service fails, Kubernetes restarts it automatically.

A screen shot of a computer program

AI-generated content may be incorrect.

* **Load balancing**: The Ingress controller distributes requests across multiple frontend pods, reducing downtime risks.

Together, these features create a **self-healing architecture** that aligns with enterprise expectations of continuous availability.

**Scalability and Performance**

Enterprise systems must adapt to changing demand:

* **Horizontal scaling**: Kubernetes allows frontend and backend replicas to be scaled up or down easily.
* **Container orchestration**: Workloads are distributed efficiently across nodes, ensuring resources are used optimally.
* **Ingress routing**: Requests are efficiently routed to the correct service, reducing latency.

This supports the non-functional requirement of **scalability** and ensures that the system can grow with user demand.

**Maintainability and Deployment Security**

A well-structured enterprise system must be maintainable and resilient against supply chain risks:

* **Containerization**: Both frontend and backend are packaged as Docker images, ensuring consistent deployments.
* **Trusted registries**: Images are stored in GitHub Container Registry with secure pull secrets.
* **Infrastructure as Code**: Kubernetes manifests define deployments, services, and ingress rules, making the system repeatable and easier to manage.

This strengthens the **maintainability and auditability** of the system, while also reducing risks during updates.

**Observability and Monitoring**

Detecting and responding to issues quickly is another non-functional concern:

* **Logs and events**: Kubernetes records events such as certificate renewals, pod restarts, and probe failures.
* **Certificate management**: Errors in TLS renewal are logged, giving administrators visibility into potential outages.
* **System health endpoints**: The /api/health endpoint provides a simple but effective way to observe backend status.

This supports the non-functional requirement of **operational transparency**.

**Conclusion**

The Talentlink project demonstrates that enterprise architecture is not only about system features but also about non-functional qualities that guarantee secure, reliable, and scalable operation. By applying Ingress with TLS, restricting backend access, adding network policies, using probes for reliability, and containerizing deployments, Talentlink was designed with security, availability, scalability, and maintainability in mind.

These decisions reflect enterprise architecture principles, ensuring the system can operate safely and efficiently in a production environment.

Learning outcome relation:  
**Learning Outcome 3: Design enterprise architectures that address non-functional requirements (security, availability, scalability, maintainability).**

**Reason why it relates:**

* The essay focuses on how Kubernetes and supporting tools (Ingress, TLS, probes, replicas, network policies) were used not to add new features, but to make the system secure, resilient, and scalable.
* By explaining liveness and readiness probes, you showed how reliability and availability are achieved in practice.
* By separating frontend (public) and backend (internal), you demonstrated architectural security principles.
* By adding TLS certificates, you addressed confidentiality and trust, which are key non-functional requirements.
* All of these are central to **enterprise architecture design**, where success is measured less by features and more by whether the system can run safely, reliably, and at scale.

**Learning Outcome 2 – Develop and deploy applications in distributed environments (Weaker but still present)**

* **Why:** You run frontend and backend in Kubernetes, expose services, and manage networking between pods.
* It’s less about coding and more about **deployment in a distributed system**, but it’s still relevant.

**Learning Outcome 5 – Ensure system quality and non-functional requirements (Moderate)**

* **Why:** Readiness and liveness probes are tied to **availability** and **resilience**, while TLS and Network Policies are tied to **security**.
* These are classic **non-functional requirements** (NFRs), and you show how they were addressed.