제22장

# 건축 위험 분석

모든 아키텍처에는 위험이 따릅니다. 운영상의 위험(가용성, 확장성, 데이터 무결성 등)도 있고, 구조적인 위험(논리적 구성 요소 간의 정적 결합 등)도 있습니다. 아키텍처 위험 분석은 아키텍트의 가장 중요한 활동 중 하나로, 이를 통해 아키텍처 내의 결함과 구조적 문제점을 파악하고 시정 조치를 취할 수 있습니다. 이 장에서는 위험을 정량화, 평가 및 식별하는 데 유용한 몇 가지 핵심 기법과 사례를 소개하고, '리스크 스토밍'이라는 활동을 살펴봅니다.

## 위험 매트릭스

아키텍처 위험을 평가할 때 가장 먼저 결정해야 할 것은 위험 수준입니다. 즉, 아키텍처의 특정 부분에 대한 위험이 낮음, 중간 또는 높음인지 판단하는 것입니다. 여기서 어려운 점은 위험 평가가 주관적일 수 있다는 것입니다. 어떤 아키텍트는 아키텍처의 특정 부분이 높은 위험이라고 생각할 수 있지만, 다른 아키텍트는 같은 부분이 중간 위험이라고 생각할 수도 있습니다. 여기서 '의견'이라는 단어를 강조하기 위해 이탤릭체로 표시했습니다. 다행히 아키텍트들은 위험을 보다 측정 가능하게 만들어주는 유용한 위험 평가 매트릭스를 가지고 있습니다.

건축 위험 평가 매트릭스 (그림 22-1) 는 위험을 평가하기 위해 두 가지 차원을 사용합니다. 즉, 관련된 위험의 전반적인 영향과 해당 위험이 발생할 가능성입니다.
건축가는 각 차원을 낮음(1), 중간(2), 높음(3)으로 평가한 다음, 매트릭스의 각 교차점에 있는 숫자를 곱합니다. 이렇게 하면 위험을 수치로 표현할 수 있어 위험을 평가하는 과정을 더욱 객관적으로 만들 수 있습니다. 1과 2는 낮은 위험(일반적으로 녹색으로 표시), 3과 4는 중간 위험(일반적으로 노란색으로 표시), 6부터 9까지는 높은 위험(일반적으로 빨간색으로 표시)으로 간주됩니다. 회색조 렌더링이나 색상을 구분하기 어려운 사람들을 위해 음영을 사용하는 것이 도움이 될 수 있습니다.

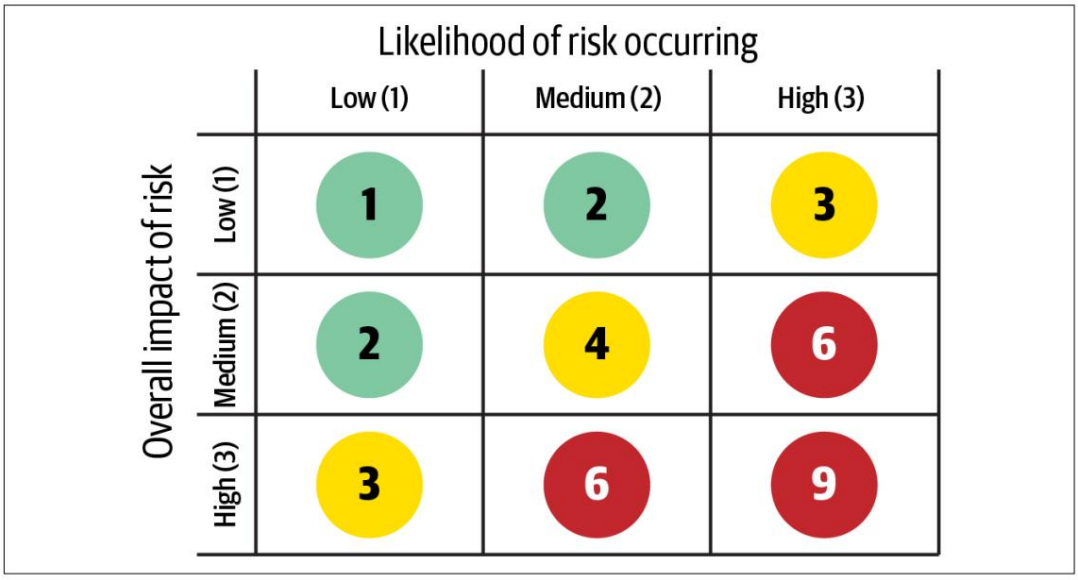


그림 22-1. 아키텍처 위험도를 판단하기 위한 매트릭스

그 유용성을 보여드리기 위해 위험 매트릭스를 예시로 사용하겠습니다. 애플리케이션의 주요 중앙 데이터베이스 가용성에 대해 우려하고 있다고 가정해 보겠습니다.

이 매트릭스를 사용하여 위험을 평가할 때는 영향을 먼저 고려하고 가능성을 두 번째로 고려하십시오. 가능성을 확신할 수 없는 경우 확인할 수 있을 때까지 높은 등급(3)을 사용하십시오.

먼저, 전반적인 영향, 즉 데이터베이스에 장애가 발생하거나 접속이 불가능해질 경우 어떤 일이 벌어질지 고려해야 합니다. 예를 들어, 영향이 심각하다고 판단하여 그림 22-1 의 매트릭스 마지막 행에 3(중간), 6(높음), 또는 9(높음)와 같은 위험 등급을 매길 수 있습니다. 하지만 두 번째 요소인 위험 발생 가능성을 고려해 보면, 데이터베이스가 클러스터 구성으로 고가용성 서버에 저장되어 있어 접속이 불가능해질 가능성이 낮다는 것을 알게 됩니다(매트릭스의 첫 번째 열). 따라서 영향은 크지만 발생 가능성은 낮은 경우, 기본 중앙 데이터베이스의 가용성에 대한 전반적인 위험 등급은 3(중간 위험)으로 결정됩니다.

## 위험 평가

리스크 매트릭스를 사용하여 리스크 평가라고 하는 것을 작성할 수 있습니다. 리스크 평가는 아키텍처의 전반적인 리스크를 요약한 보고서로, 컨텍스트(서비스, 하위 도메인 영역 또는 시스템의 도메인 영역)에 기반한 의미 있는 평가 기준을 포함합니다. 저희는 수많은 리스크 평가를 수행했으며, 아키텍처 특성이 매우 우수하다는 것을 확인했습니다.

위험 평가 기준. 시스템의 핵심 아키텍처 특성이 확장성, 탄력성 및 데이터 무결성인데 왜 성능 위험을 분석하는 데 시간을 낭비해야 할까요?
4장 에서 설명한 것과 같은 특징들을 파악하는 것이 건축 위험을 분석하는 첫 번째 단계입니다.

건축물이 지탱해야 할 가장 중요한 건축적 특징들은 훌륭한 위험 평가 기준이 됩니다.

위험 평가 보고서의 기본 형식은 그림 22-2에 나와 있습니다. 여기서 1과 2는 낮은 위험을, 3과 4는 중간 위험을, 6과 9는 높은 위험을 나타냅니다.
스프레드시트의 왼쪽에는 위험 기준이, 위쪽에는 상황 정보가 나열되어 있습니다.

| RISK CRITERIA | Customer registration | Catalog checkout | Order fulfillment | Order shipment | TOTAL RISK |
|---|---|---|---|---|---|
| Scalability | 2 | 6 | 1 | 2 | 11 |
| Availability | 3 | 4 | 2 | 1 | 10 |
| Performance | 4 | 2 | 3 | 6 | 15 |
| Security | 6 | 3 | 1 | 1 | 11 |
| Data integrity | 9 | 6 | 1 | 1 | 17 |
| TOTAL RISK | 24 | 21 | 8 | 11 | |

그림 22-2. 표준 위험 평가의 예

본 위험 평가에서는 전자상거래 주문 시스템을 예시로 사용합니다. 시스템의 핵심 아키텍처적 특성을 나타내는 다섯 가지 기준을 평가합니다.
상단에는 각각 별도의 도메인(고객 등록, 카탈로그 결제, 주문 처리, 주문 배송)을 나타내는 네 가지 컨텍스트가 있습니다. 도메인 또는 하위 도메인 컨텍스트를 사용하는 것이 효과적입니다. 서비스 수준에서 위험을 분석하는 것은 일반적으로 너무 세분화되어 여러 서비스 간의 통신 또는 조정과 관련된 위험을 고려하지 못하기 때문입니다.

정량화된 위험을 사용하는 장점은 위험 기준과 맥락을 모두 고려한다는 점입니다. 예를 들어, 그림 22-2 에서 데이터에 대한 총 누적 위험은 다음과 같습니다.

무결성 점수는 17점으로, 기준 관점에서 가장 위험도가 높은 영역입니다. 가용성의 누적 위험도는 10점으로, 기준상 가장 낮은 위험도입니다. 하지만 각 컨텍스트 영역의 상대적 위험도를 비교해 보면, 고객 등록이 컨텍스트 측면에서 가장 위험도가 높은 영역이고, 주문 처리가 가장 위험도가 낮은 영역입니다. 이러한 정보는 우선순위를 정하고 위험 감소를 위해 추가적인 노력을 기울여야 할 부분을 결정하는 데 유용합니다.

이 위험 평가 예시에는 모든 위험 분석 결과가 포함되어 있지만, 특정 문제를 강조하기 위해 세부 정보를 필터링하는 것이 유용할 때가 있습니다. 예를 들어, 시스템 설계자인 당신이 이해관계자들에게 시스템의 고위험 영역에 대해 발표하는 회의에 참석했다고 가정해 보겠습니다. 그림 22-2 처럼 전체 위험 평가를 제시하는 대신 , 저위험 및 중간 위험 영역(잡음)을 필터링하여 고위험 영역(신호)을 강조할 수 있습니다. 신호 대 잡음비를 개선하면 더 효과적이고 혼란을 줄이는 메시지를 전달할 수 있습니다. 그림 22-3은 동일한 위험 평가의 필터링된 버전을 보여줍니다. 두 이미지를 비교하여 필터링된 평가가 메시지를 얼마나 더 명확하게 전달하는지 확인해 보세요.

| RISK CRITERIA | Customer registration | Catalog checkout | Order fulfillment | Order shipment | TOTAL RISK |
|---|---|---|---|---|---|
| Scalability | | 6 | | | 6 |
| Availability | | | | | 0 |
| Performance | | | | 6 | 6 |
| Security | 6 | | | | 6 |
| Data integrity | 9 | 6 | | | 15 |
| TOTAL RISK | 15 | 12 | 0 | 6 | |

그림 22-3. 위험 평가에서 높은 위험만 표시하도록 필터링하기

위험 평가의 전체 버전에는 한 가지 문제점이 있는데, 그것은 특정 시점의 상황만 보여줄 뿐, 상황이 개선되고 있는지 악화되고 있는지를 보여주지 않는다는 것입니다. 다시 말해, 그림 22-2는 위험의 방향을 보여주지 않습니다. 6 장 에서 설명한 적합도 함수를 통한 연속적인 측정을 사용하면 위험의 방향을 파악할 수 있습니다. 각 위험 기준을 객관적으로 분석하면 추세를 관찰하여 각 위험 기준의 방향을 확인할 수 있습니다.

그림 22-4 에서는 위험 평가에 세 번째 차원인 방향을 추가합니다. 특정 기준과 상황에 대한 위험이 악화되고 있음을 나타내기 위해 바로 선 삼각형을 사용합니다. 삼각형의 꼭짓점이 위쪽, 즉 더 높은 숫자를 향하도록 합니다. 반대로, 위험이 감소하고 있음을 나타내기 위해 거꾸로 된 삼각형을 사용합니다(즉, 꼭짓점이 아래쪽, 즉 더 낮은 숫자를 향하도록 합니다). 마지막으로, 위험이 변하지 않고 더 나아지지도, 더 나빠지지도 않는 상태를 나타내기 위해 원을 사용합니다. 이러한 방향 표시는 혼란스러울 수 있으므로, 방향을 나타내는 기호를 사용할 때는 항상 범례를 포함하는 것이 좋습니다.

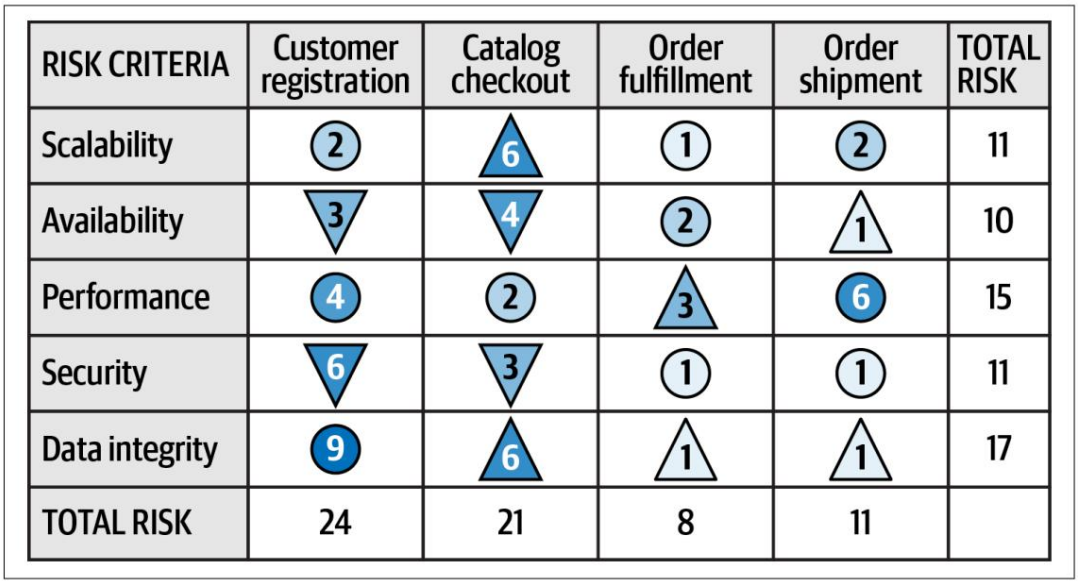


그림 22-4. 삼각형을 이용하여 위험 방향을 나타낸 그림

방향성을 보여주는 이번 수정된 아키텍처 위험 평가에서는 기존 평가와는 다른 양상을 확인할 수 있습니다. 첫째, 카탈로그 결제, 주문 처리 및 배송 과정에서 데이터 무결성이 지속적인 측정 결과(위쪽을 가리키는 삼각형)를 통해 악화되고 있는 것으로 나타나 데이터베이스 문제일 가능성을 시사합니다. 하지만 고객 등록 및 카탈로그 결제 과정에서는 보안 및 가용성이 전반적으로 개선되고 있는 것(아래쪽을 가리키는 삼각형)으로, 해당 영역의 개선을 보여줍니다.

다음으로, 팀이 특정 상황과 기준에 대한 위험 수준을 파악하는 데 사용할 수 있는 '위험 폭풍'이라는 프로세스를 설명합니다.

# 위험 폭풍

두 가지 이유 때문에 어떤 아키텍트도 시스템의 전반적인 위험을 혼자서 판단할 수는 없습니다. 첫째, 혼자 작업하는 아키텍트는 위험 영역을 놓치거나 간과할 수 있습니다. 둘째, 시스템의 모든 부분을 완벽하게 알고 있는 아키텍트는 거의 없습니다. 바로 이 부분에서 리스크 스토밍이 도움이 될 수 있습니다.

리스크 스토밍은 특정 차원(컨텍스트 또는 기준) 내에서 아키텍처 리스크를 파악하기 위한 협업 활동입니다. 대부분의 리스크 스토밍에는 여러 명의 아키텍트가 참여하지만, 시니어 개발자와 테크 리더도 함께 참여시키는 것을 강력히 권장합니다. 이들은 아키텍처 리스크에 대한 구현 관점을 제공할 뿐만 아니라, 참여를 통해 아키텍처를 더 잘 이해할 수 있도록 도와줍니다.

리스크 스토밍은 식별, 합의, 완화의 세 단계로 구성됩니다. 개별 단계(1단계)에서는 모든 참가자가 각자 독립적으로 리스크 매트릭스를 사용하여 아키텍처의 다양한 영역에 리스크를 할당합니다. 이 개별 단계는 참가자들이 서로에게 영향을 미치거나 특정 영역에서 주의를 분산시키지 않도록 하는 데 필수적입니다. 협업 단계인 2단계에서는 모든 참가자가 함께 리스크 영역에 대한 합의를 도출하고 논의하며(2단계), 리스크를 완화하기 위한 해결책을 마련합니다(3단계).

세 단계 모두 포괄적 또는 맥락적 아키텍처 다이어그램을 활용합니다( 23 장 참조 ). 위험 요소 분석 회의를 진행하는 아키텍트(이하 진행자)는 위험 요소 분석 회의에 필요한 최신 다이어그램을 모든 참가자에게 제공해야 합니다.

그림 22-5는 위험 분석 프로세스를 설명하기 위해 사용할 예시 아키텍처를 보여줍니다. 이 아키텍처에서 Elastic Load Balancer는 웹 서버(Nginx)와 애플리케이션 서비스가 포함된 각 EC2 인스턴스로 요청을 전달합니다. 애플리케이션 서비스는 MySQL 데이터베이스, Redis 캐시, 그리고 MongoDB 데이터베이스(로깅용)를 호출합니다. 또한 푸시 확장 서버(Push Expansion Server)를 호출하는데, 이 서버들은 다시 MySQL 데이터베이스, Redis 캐시, MongoDB 로깅 기능과 상호 작용합니다.

(이러한 제품이나 전문 용어를 모두 이해하지 못하더라도 걱정하지 마세요. 이처럼 지나치게 모호하고 일반화된 구조는 위험 관리 기법이 어떻게 작동하는지 설명하기 위한 예시일 뿐입니다.)

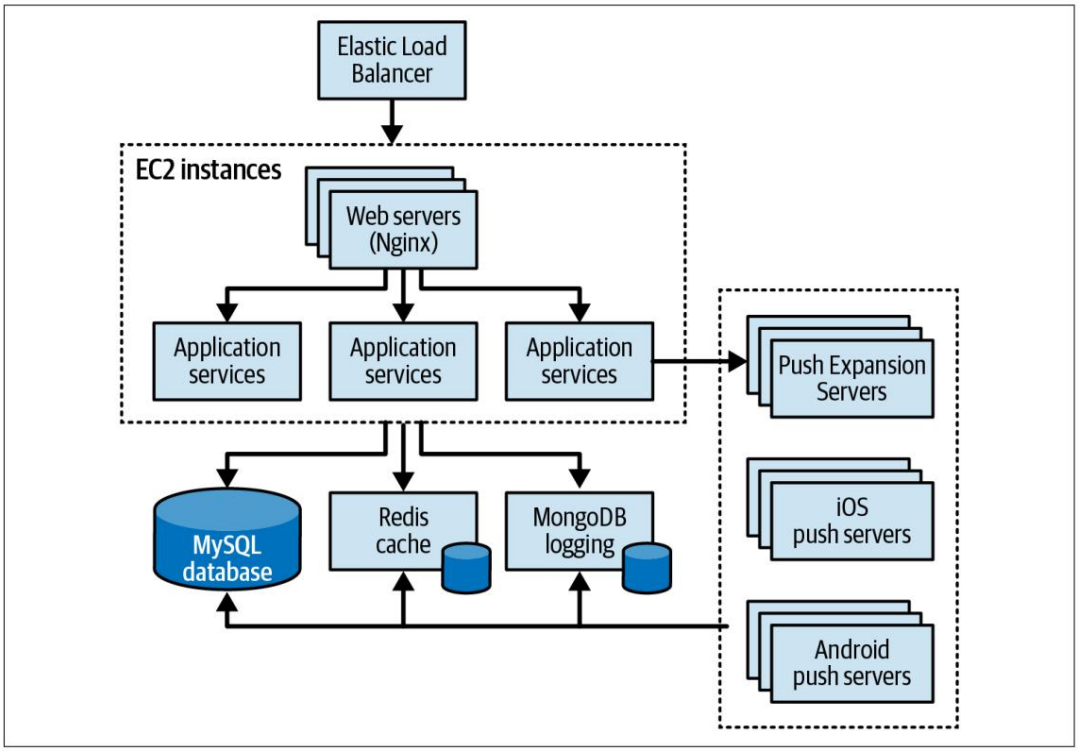


그림 22-5. 위험 고찰 회의를 위한 아키텍처 다이어그램 예시

1단계부터 시작하겠습니다.

## 1단계: 식별

리스크 스토밍의 식별 단계에서는 각 참가자가 아키텍처 내의 위험 영역을 개별적으로 식별합니다. 이 단계에서 각 참가자는 다른 참가자의 의견에 영향을 받거나 좌우되지 않고 관련된 위험에 대한 자신의 객관적인 견해를 기록하는 것이 매우 중요합니다. 식별 단계는 세 단계로 구성됩니다.

1. 진행자는 모든 참가자에게 협업 단계 초대장을 보냅니다. 초대장에는 아키텍처 다이어그램(또는 다이어그램을 찾을 수 있는 위치), 분석할 위험 기준 및 맥락, 협업 세션의 날짜, 시간 및 장소(실제 또는 가상)와 기타 물류 관련 세부 정보가 포함됩니다.

2. 참가자들은 위험 매트릭스를 사용하여 아키텍처 위험을 개별적으로 분석합니다.
3. 참가자들은 각 위험도를 낮음(1~2), 중간(3~4), 높음(6~9)으로 분류하고, 해당 숫자를 작은 초록색, 노란색 또는 빨간색 포스트잇에 적습니다.

대부분의 위험 분석 활동은 특정 기준이나 맥락(예: "보안 위험은 어디에 있는가?" 또는 "고객 등록 과정에서 위험에 처한 영역은 무엇인가?")만을 분석합니다. 그러나 직원 가용성이나 시간적 제약이 있는 경우,

위험 평가팀은 특정 맥락(예: 성능 및 확장성) 내에서 여러 차원을 분석해야 할 수 있습니다. 이러한 경우 참가자들은 일반적으로 포스트잇에 위험 번호 옆에 구체적인 기준을 적습니다. 예를 들어, 세 명의 참가자가 중앙 데이터베이스와 관련된 위험을 식별했다고 가정해 보겠습니다. 세 참가자 모두 위험을 높음(6)으로 식별했지만, 한 참가자는 이를 가용성에 대한 위험으로 보고 나머지 두 참가자는 성능에 대한 위험으로 봅니다. 참가자들은 이 두 가지 기준에 대해 별도로 논의해야 합니다.

가능하면 위험 분석 활동을 단일 기준 또는 맥락으로 제한하십시오. 이렇게 하면 참가자들이 이 특정 차원에 집중할 수 있고 실제 위험이 무엇인지에 대한 혼란을 방지할 수 있습니다.

## 2단계: 합의 도출

리스크 브레인스토밍의 합의 도출 단계는 매우 협력적인 과정으로, 아키텍처 내의 리스크에 대해 모든 참여자 간의 합의를 얻는 것을 목표로 합니다.
이 활동은 진행자가 벽에 큰 인쇄된 아키텍처 다이어그램(또는 큰 화면에 전자 버전)을 게시할 때 가장 효과적입니다. 참가자들이 위험 브레인스토밍 세션에 도착하면 진행자는 참가자들에게 아키텍처 다이어그램의 관련 영역에 위험 수준을 적은 포스트잇을 붙이도록 지시합니다( 그림 22-6 참조).

모든 포스트잇이 제자리에 놓이면 협업 단계가 시작됩니다. 이 단계의 목표는 팀으로서 위험 영역을 분석하고 위험 수준에 대한 합의에 도달하는 것입니다.
그림 22-6 에 나타난 예시에서 팀은 여러 위험 영역을 파악했습니다.
(실제 기준은 이 예시에서 중요하지 않습니다.) 다음과 같은 사실을 알 수 있습니다.

- 두 참가자는 Elastic Load Balancer를 중간 위험으로 인식했습니다(3).
  한 참가자는 이를 고위험으로 식별했습니다(6).
- 한 참가자는 푸시 확장 서버를 고위험군으로 식별했습니다(9). • 세 참가자는 MySQL 데이터베이스를 중간 위험군으로 식별했습니다(3). • 한 참가자는 Redis 캐시를 고위험군으로 식별했습니다(9). • 세 참가자는 MongoDB 로깅을 저위험군으로 식별했습니다(2). • 아키텍처의 다른 영역에 위험을 초래하는 요소를 식별한 참가자는 없었으므로 다른 영역에 대한 포스트잇 메모는 없습니다.

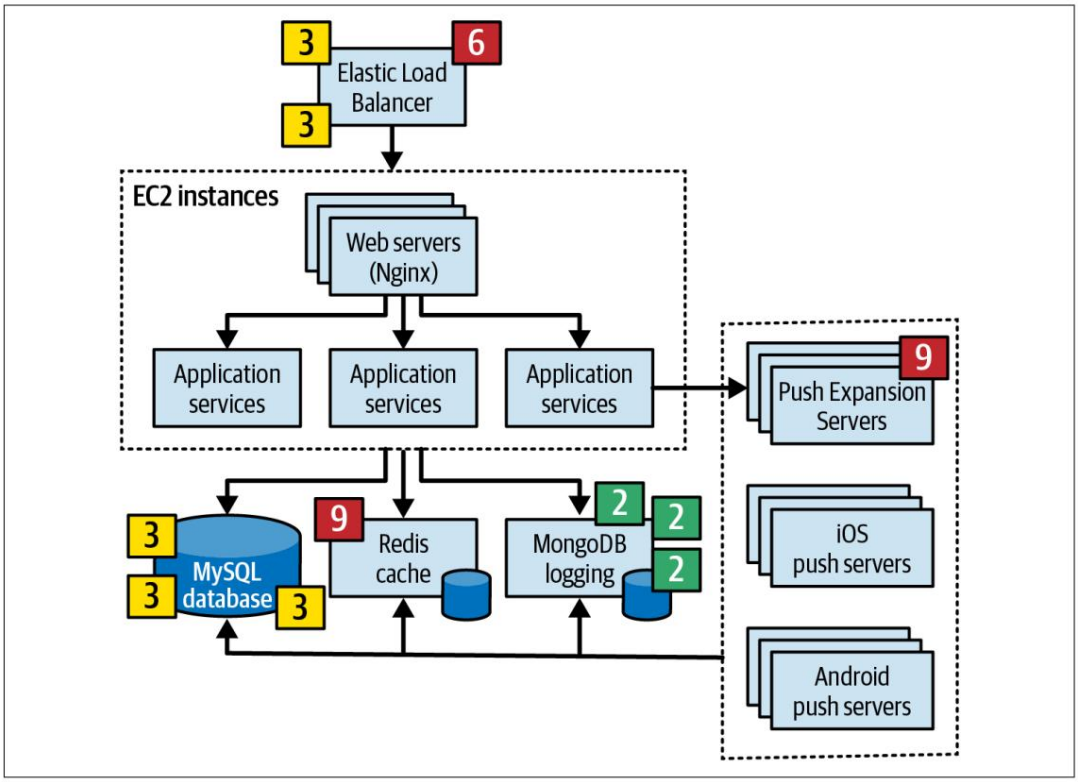


그림 22-6. 위험 지역의 초기 식별

MySQL 데이터베이스와 MongoDB 로깅은 모든 참가자가 위험 수준에 동의했으므로 이번 회의에서 더 이상 논의할 필요가 없습니다. 하지만 Elastic Load Balancer에 대해서는 의견 차이가 있으며, Push Expansion Server와 Redis 캐시는 각각 한 명의 참가자만 위험 요소로 인식했습니다. 이러한 의견 차이를 해소하는 것이 바로 협업 단계의 핵심입니다.

두 참가자(Austen과 Logan)는 Elastic Load Balancer를 중간 위험(3)으로 식별했고, 한 참가자(Addison)는 높은 위험(6)으로 식별했습니다. Austen과 Logan은 Addison에게 왜 위험을 높게 평가했는지 묻습니다. Addison은 Elastic Load Balancer가 다운되면 전체 시스템에 접근할 수 없게 된다고 답합니다. 이는 사실이며 영향 등급을 높게 만드는 요인이지만, 다른 두 참가자는 클러스터링으로 인해 이러한 일이 발생할 위험이 낮다고 Addison을 설득합니다. Addison은 이에 동의하고, 그룹은 발생 가능성 위험 수준을 중간(3)으로 낮춥니다.

하지만 상황은 다르게 흘러갈 수도 있었습니다. 만약 오스틴과 로건이 애디슨이 발견한 엘라스틱 로드 밸런서의 특정 위험 요소를 놓쳤다면, 애디슨은 다른 두 참가자를 설득하여 해당 위험 수준을 중간이 아닌 높음으로 분류하도록 했을지도 모릅니다. 이것이 바로 위험 분석 과정에서 협업 단계가 매우 중요한 이유입니다.

한 참가자는 푸시 확장 서버를 고위험군으로 식별했지만(9), 다른 참가자는 아키텍처의 이 영역에서 어떠한 위험도 식별하지 못했습니다. 위험을 식별한 참가자는 이 아키텍처의 부하와 유사한 높은 부하에서 푸시 확장 서버가 지속적으로 다운되는 좋지 않은 경험을 했기 때문에 위험을 높게 평가했다고 설명합니다. 이 예는 위험 폭풍의 가치를 보여줍니다. 해당 참가자의 참여가 없었다면 아무도 프로덕션 단계가 한참 진행될 때까지 고위험을 인지하지 못했을 것입니다.

Redis 캐시는 흥미로운 사례입니다. Devon이라는 개발자인 한 참가자는 이를 고위험으로 식별했지만(9), 다른 누구도 해당 캐시에 위험이 있다고 보지 않았습니다.
다른 참가자들이 Devon에게 이 위험을 높게 평가한 이유를 묻자 Devon은 “Redis 캐시가 뭐예요?”라고 대답합니다. 위험 논의 참가자가 자신이 알지 못하는 기술을 식별할 때마다 해당 영역에는 자동으로 높은 위험 수준이 할당됩니다(9).

검증되지 않았거나 알려지지 않은 기술에는 항상 가장 높은 위험 등급(9)을 부여합니다. 위험 매트릭스는 이 기준이나 맥락에 사용할 수 없기 때문입니다.

Redis 캐시 사례는 개발자를 위험 분석 회의에 참여시키는 것이 왜 중요한지 보여줍니다. 참가자가 특정 기술에 대해 알지 못한다는 사실은 아키텍트에게 전반적인 위험에 대한 귀중한 정보를 제공합니다. 아키텍트는 이를 바탕으로 기술을 변경하거나 개발팀의 역량을 향상시키기 위한 교육 비용을 투자할 수 있습니다.

이 단계는 모든 참가자가 식별된 위험 영역에 동의할 때까지 계속됩니다. 모든 포스트잇이 정리되면 이 단계가 종료됩니다. 최종 결과는 그림 22-7에 나와 있습니다.

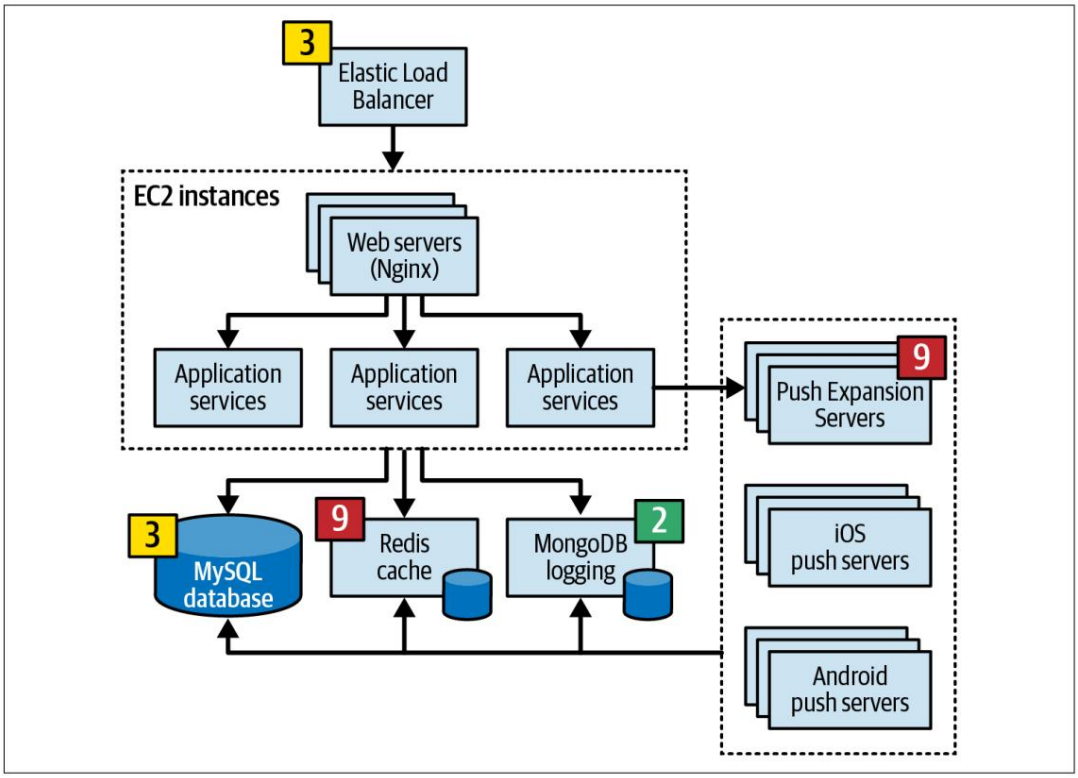


그림 22-7. 위험 영역에 대한 합의

3단계: 위험 완화 모든 참여자가

아키텍처의 위험 수준에 동의하면 위험 완화 단계가 시작됩니다. 위험 완화는 일반적으로 완벽하다고 여겨졌던 아키텍처의 특정 영역을 변경하는 것을 포함합니다.

이 단계 역시 협업을 통해 진행되며, 2단계에서 파악된 위험 요소를 줄이거나 제거하는 방법을 모색합니다. 파악된 위험 요소에 따라 기존 아키텍처를 완전히 변경해야 할 수도 있고, 특정 영역에서 간단한 아키텍처 리팩토링만 수행하여 처리량 병목 현상을 줄이는 것(예: 백프레셔용 큐 추가)으로 제한할 수도 있습니다.

어떤 변경 사항이 필요하든 위험 완화 단계에는 일반적으로 추가 비용이 발생합니다. 따라서 이 단계에는 특정 완화 솔루션의 비용이 위험보다 큰지 여부를 결정할 권한을 가진 주요 비즈니스 이해관계자가 참여하는 것이 중요합니다.

예를 들어, 앞서 예시로 든 위험 논의 세션에서 팀이 중앙 데이터베이스가 전체 시스템 가용성과 관련하여 중간 수준의 위험(4)을 가지고 있다고 판단했다고 가정해 보겠습니다. 참가자들은 데이터베이스를 클러스터링하고 이를 여러 개의 물리적 데이터베이스로 분리하면 해당 위험을 완화할 수 있다는 데 동의합니다. 그러나 이러한 해결책에는 비용이 발생합니다.

5만 달러. 담당 아키텍트는 소유주를 포함한 주요 비즈니스 이해관계자들과 만나 가용성 위험과 비용 간의 장단점을 논의합니다. 소유주는 가격이 너무 높고 가용성 위험을 감수할 가치가 없다고 판단합니다. 그러자 아키텍트는 다른 접근 방식을 제안합니다. 비용이 많이 드는 클러스터링 대신 데이터베이스를 두 개의 도메인 기반 데이터베이스로 분리하는 것은 어떻겠냐는 것입니다. 이 솔루션은 1만 6천 달러의 비용으로 가용성 위험을 줄일 수 있습니다.

이해관계자들은 이러한 타협안에 동의합니다.

이 시나리오는 리스크 스토밍이 전체적인 아키텍처뿐만 아니라 아키텍트와 비즈니스 이해관계자 간의 협상에도 어떤 영향을 미치는지 보여줍니다. 이 장의 서두에서 설명한 리스크 평가와 결합된 리스크 스토밍은 리스크를 식별하고 추적하며, 아키텍처를 개선하고, 주요 이해관계자 간의 협상을 구조화하는 데 매우 효과적인 도구입니다.

## 사용자 스토리 위험 분석

리스크 스토밍은 아키텍처 리스크 식별 외에도 소프트웨어 개발의 여러 측면에서 유용합니다. 예를 들어, 개발팀은 스토리 그루밍 과정에서 특정 스프린트 내 사용자 스토리 완료와 관련된 전반적인 리스크(그리고 결과적으로 해당 스프린트의 전반적인 리스크 평가)를 파악하기 위해 리스크 스토밍을 활용할 수 있습니다. 동일한 리스크 매트릭스를 사용하여, 팀은 해당 스프린트 내에 스토리가 완료되지 않을 경우 발생할 수 있는 전반적인 영향과 현재 스프린트에서 스토리가 완료되지 않을 가능성을 파악함으로써 사용자 스토리의 리스크를 식별할 수 있습니다. 그런 다음 위험도가 높은 스토리를 선별하고, 면밀히 추적하며, 우선순위를 효과적으로 지정할 수 있습니다.

## 위험 분석 활용 사례

리스크 스토밍의 효과와 그것이 전반적인 아키텍처를 어떻게 개선할 수 있는지 설명하기 위해, 간호사들이 환자들에게 다양한 건강 상태에 대해 조언하는 콜센터 지원 시스템을 예로 들어보겠습니다. 시스템 요구사항은 다음과 같습니다.

- 제3자 진단 엔진이 간호사와 환자에게 의료 문제에 대한 질문을 제시하고 안내합니다. 이 엔진은 초당 약 500건의 요청을 처리할 수 있습니다.

- 환자는 콜센터에 전화하여 간호사와 상담하거나, 동일한 진단 엔진에 직접 접속할 수 있는 셀프 서비스 웹사이트를 이용할 수 있습니다.
- 이 시스템은 전국적으로 최대 250명의 간호사가 동시에 업무를 처리할 수 있어야 하며, 수십만 명의 환자가 동시에 셀프 서비스 방식으로 진료를 받을 수 있어야 합니다. • 간호사는 의료 기록

교환 시스템을 통해 환자의 의료 기록에 접근할 수 있지만, 환자는 자신의 의료 기록에 접근할 수 없습니다.

- 시스템은 HIPAA (건강보험 이동성 및 책임법)를 준수해야 합니다. 즉, 간호사 외에는 누구도 환자의 의료 기록에 접근할 수 없어야 합니다. 셀프 서비스 방식은 HIPAA 준수를 보장할 수 없습니다. • 시스템은 감기, 독감 및 코로나19 유행 기간 동안 발생하는 높은 처리량을 처리할 수 있어야 합니다.

  발병.

- 전화는 각 간호사의 기술 프로필(예: 구사 언어 또는 의료 전문 분야)에 따라 간호사에게 연결됩니다.

이러한 요구사항을 분석한 후, 이 시스템을 담당하는 아키텍트인 로건은 그림 22-8 에 나타낸 고수준 아키텍처를 설계했습니다 . 이 아키텍처는 세 개의 독립적인 웹 기반 사용자 인터페이스(UI)로 구성됩니다. 하나는 셀프 서비스용, 하나는 전화를 받는 간호사용, 그리고 나머지 하나는 관리 직원이 간호사 프로필과 구성 설정을 추가 및 유지 관리하는 데 사용하는 UI입니다. 시스템의 콜센터 부분은 전화를 받는 콜 접수 서비스와 간호사의 기술 프로필을 기반으로 발신자를 다음으로 이용 가능한 간호사에게 연결하는 콜 라우터 서비스로 구성됩니다. 콜 라우터 서비스는 중앙 데이터베이스에 접근하여 간호사 프로필 정보를 가져옵니다. 이 아키텍처의 핵심은 보안 검사를 수행하고 요청을 적절한 백엔드 서비스로 전달하는 진단 시스템 API 게이트웨이입니다.

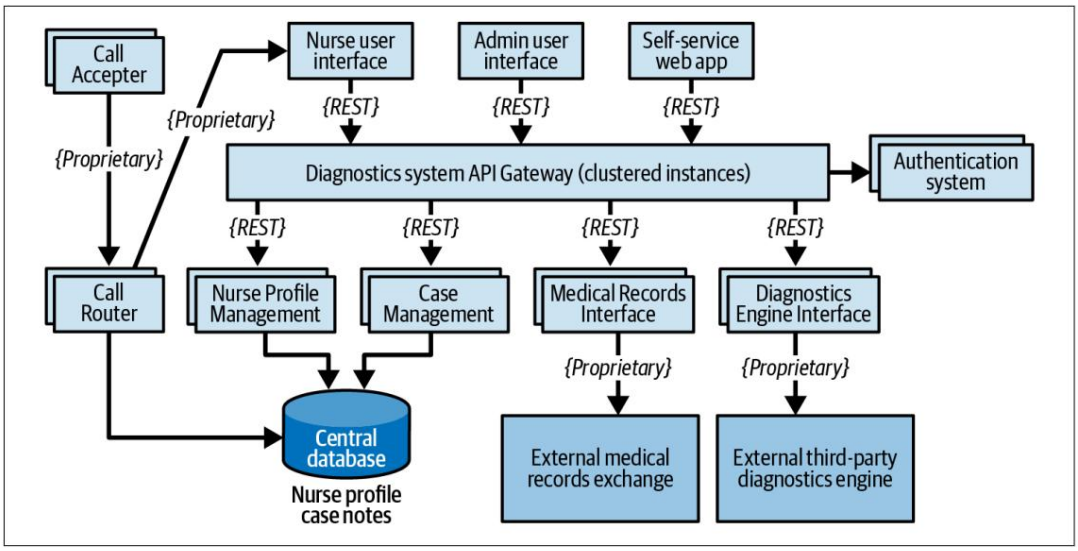


그림 22-8. 간호 상담 전화 진단 시스템의 고수준 아키텍처

이 시스템의 네 가지 주요 서비스는 사례 관리 서비스, 간호사 프로필 관리 서비스, 의료 기록 교환 시스템과의 의료 기록 인터페이스 서비스, 그리고 외부 제3자 진단 엔진 인터페이스 서비스입니다. 외부 시스템 및 콜센터 서비스와의 통신은 자체 프로토콜을 사용하는 경우를 제외하고 모든 통신은 REST API를 사용합니다. 아키텍처가 지원해야 하는 핵심 영역은 가용성, 확장성 및 보안입니다.

여러 차례 검토 끝에 로건은 해당 아키텍처가 구현 준비가 완료되었다고 판단했습니다.
하지만 책임감 있고 유능한 건축가인 로건은 위험 요소 분석 회의를 개최하기로 결정합니다.

진행자 로서 로건

은 첫 번째 위험 분석 활동을 시스템 성공에 매우 중요한 가용성에 집중하기로 결정합니다. 식별 및 협업 단계를 거친 후, 참가자들은 다음과 같은 위험 영역을 도출합니다( 그림 22-9 참조).

- 중앙 데이터베이스 가용성: 필요할 때 데이터베이스를 사용할 수 없을 가능성이 높고(2) 영향이 크기 때문에 위험도가 높음(6)
- 진단 엔진 가용성: 높은 영향(3)과 가용성이 없을 가능성(3)이 알려지지 않았기 때문에 위험도가 높음(9).
- 의료 기록 인터페이스 가용성: 낮은 위험(2); 이 구성 요소는 특정 의료 결과를 결정하는 데 필요하지 않습니다. • 아키텍처에는 각 서비스의 여러 인스턴스가 포함

  되어 있고 API 게이트웨이가 클러스터링되어 있으므로 팀은 시스템의 다른 부분이 가용성 위험이라고 판단하지 않았습니다.

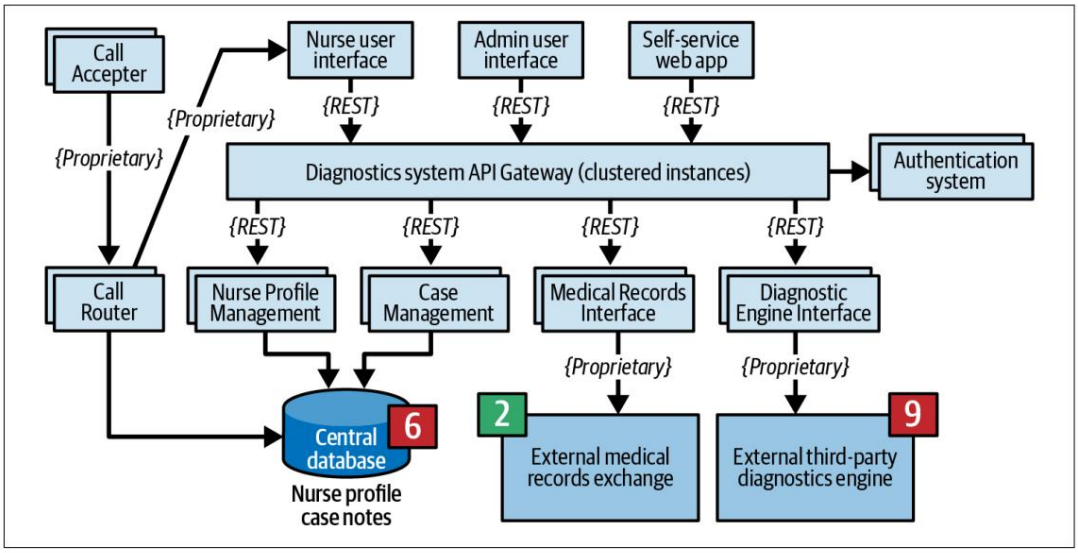


그림 22-9. 위험 분석팀이 파악한 가용성 위험 영역

모든 참가자는 데이터베이스가 다운될 경우 간호사가 수동으로 환자 기록을 작성할 수는 있지만, 통화 연결 시스템은 작동하지 않을 것이라는 데 동의했습니다. 이러한 위험을 완화하기 위해, 그들은 하나의 물리적 데이터베이스를 두 개의 독립적인 데이터베이스로 분리하기로 공동으로 결정했습니다. 하나는 간호사 프로필 정보를 포함하는 클러스터형 데이터베이스이고, 다른 하나는...

진료 기록을 위한 단일 인스턴스 데이터베이스를 사용합니다. 이러한 아키텍처 변경은 데이터베이스 가용성 문제를 해결할 뿐만 아니라 진료 기록 보안에도 도움이 됩니다.

외부 시스템(이 경우 진단 엔진 및 의료 기록 인터페이스) 은 제3자가 관리하기 때문에 가용성 위험을 완화하기가 훨씬 더 어렵습니다. 따라서 팀은 이러한 시스템에 서비스 수준 계약(SLA) 또는 서비스 수준 목표(SLO)가 게시되어 있는지 조사하기로 결정합니다. SLA는 일반적으로 법적 구속력이 있는 계약이지만, SLO는 일반적으로 법적 구속력이 없습니다.

그들은 두 시스템 모두에 대한 SLA를 마련했습니다. 진단 엔진 SLA는 99.99%의 가용성(연간 52.60분의 다운타임)을 보장하고, 의료 기록 인터페이스 SLA는 99.90%의 가용성(연간 8.77시간의 다운타임)을 보장합니다.

이 분석을 바탕으로, 위험 평가팀은 해당 정보를 통해 식별된 위험을 제거하기에 충분한 근거를 확보했습니다.

이 위험 분석 세션 후, 팀은 그림 22-10 에 나타낸 것처럼 아키텍처를 변경하여 두 개의 데이터베이스를 생성하고 아키텍처 다이어그램에 SLA를 추가합니다.

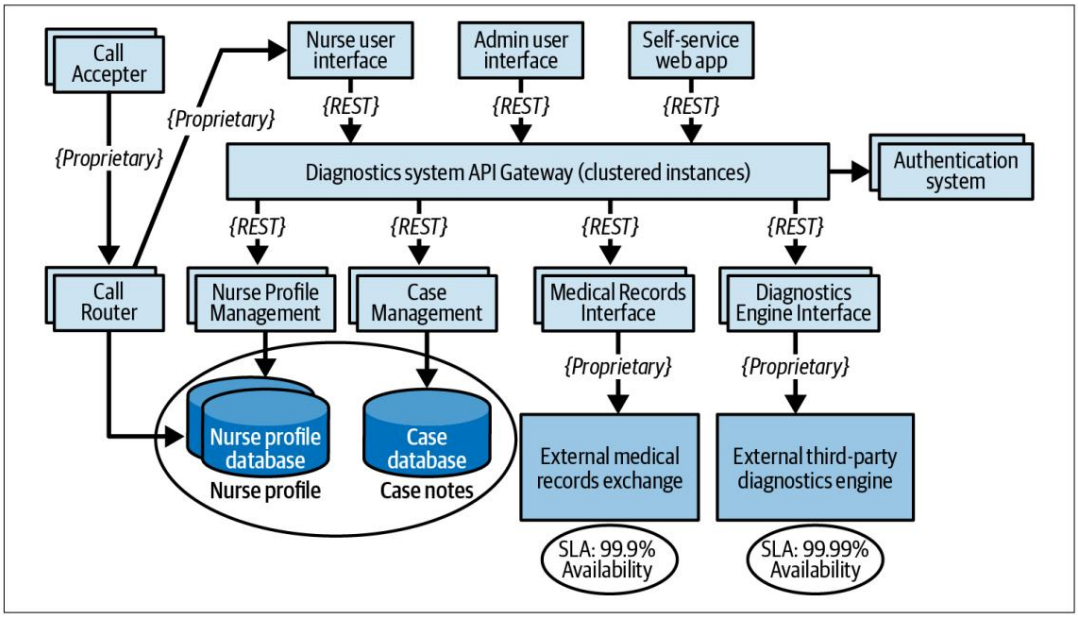


그림 22-10. 별도의 데이터베이스를 사용하면 가용성 위험 영역을 완화할 수 있습니다.

탄력성 두 번째

위험 분석 활동은 탄력성, 즉 사용자 부하 급증(가변 확장성이라고도 함)에 초점을 맞춥니다. 간호사 수는 250명에 불과하지만 (즉, 진단 엔진에 접속하는 간호사는 250명을 넘지 않음), 시스템의 셀프 서비스 부분 도 진단 엔진 에 접근할 수 있으므로 진단 엔진 인터페이스 에 대한 요청 수가 크게 증가합니다 . 위험 분석 참가자들은 독감 유행과 코로나19 발생으로 인해 시스템 부하가 크게 증가할 것을 우려하고 있습니다.

참가자들은 진단 엔진 인터페이스가 위험도가 높다고 만장일치로 판단했습니다(9). 초당 500개의 요청만 처리할 수 있기 때문에 특히 REST를 인터페이스 프로토콜로 사용하는 경우 예상되는 처리량을 따라가지 못할 위험이 높다고 정확하게 계산했습니다.

팀은 이러한 위험을 완화하는 한 가지 방법으로 API 게이트웨이와 진단 엔진 인터페이스 간의 통신에 비동기 큐(메시징)를 사용하는 것을 결정했습니다. 이는 진단 엔진 호출이 누적될 경우를 대비한 백프레셔 지점을 제공합니다 . 이는 좋은 방법이지만, 모든 위험을 완전히 제거하는 것은 아닙니다. 간호사와 셀프 서비스 환자는 여전히 진단 엔진 의 응답을 너무 오래 기다려야 하고 , 요청 시간이 초과될 가능성이 높습니다.

참가자들은 하나의 메시지 채널 대신 두 개의 메시지 채널을 사용하여 이러한 요청들을 분리하는 ' 앰뷸런스 패턴'을 사용하기로 결정했습니다. 이렇게 하면 시스템이 셀프 서비스 요청보다 간호사 요청을 우선 처리할 수 있습니다. 이는 위험을 완화하는 데 도움이 되지만 대기 시간 문제는 여전히 해결되지 않습니다. 추가 논의 끝에, 그룹은 특정 진단 질문을 캐싱하여 진단 엔진 인터페이스에 도달하지 않도록 함으로써 감염병 발생 관련 문의를 진단 엔진 으로 전송되는 횟수를 줄이기로 결정했습니다 .

팀은 간호사용과 환자 자가 진단용, 두 개의 메시지 채널을 구축하는 것 외에도 특정 발병 또는 독감 관련 질문에 대한 모든 요청을 처리하는 '진단 발병 캐시 서버' 라는 새로운 서비스를 개발했습니다 (그림 22-11). 이 새로운 아키텍처는 진단 엔진 호출 횟수를 줄여 다른 증상과 관련된 요청을 더 많이 동시에 처리할 수 있도록 합니다.

## 위험 분석 노력이 없었다면, 이 위험은 독감이 발생할 때까지 파악되지 않았을 수도 있습니다. 계절.

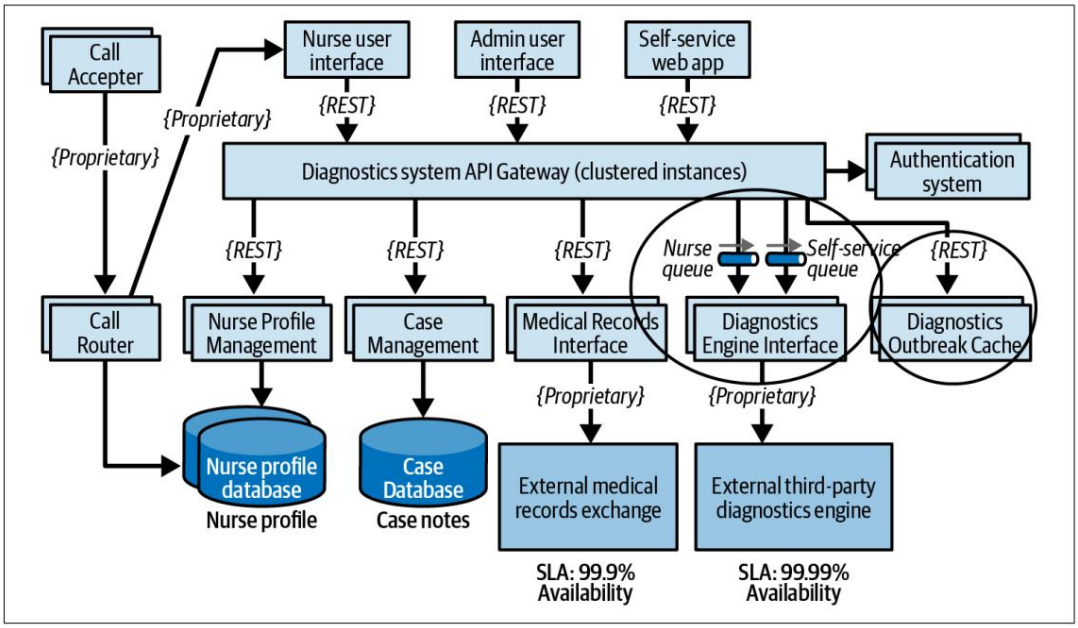


그림 22-11. 탄력성 위험을 해결하기 위한 아키텍처 수정

보안 이러

한 성공에 고무된 아키텍트는 시스템의 또 다른 중요한 특징인 보안에 초점을 맞춘 최종 위험 분석 세션을 진행하기로 결정합니다. HIPAA 규정 요건에 따라 의료 기록 인터페이스는 간호사만 환자의 의료 기록에 접근할 수 있도록 해야 합니다. 아키텍트는 API 게이트웨이의 인증 및 권한 확인 절차가 이러한 위험을 제거한다고 생각하지만, 참가자들이 다른 보안 위험을 발견할지 궁금해합니다.

참가자들은 모두 진단 시스템의 API 게이트웨이를 높은 보안 위험 요소로 인식했습니다(6). 그들은 관리 직원이나 셀프 서비스 환자가 의료 기록에 접근할 경우 미치는 영향이 크다고 언급했지만(3), 그 가능성은 중간 정도로 평가했습니다(2). 각 API 호출에 대한 보안 검사가 도움이 되기는 하지만, 모든 호출(셀프 서비스, 관리, 간호사)은 여전히 동일한 API 게이트웨이를 거칩니다. 결국 참가자들은 처음에 이 위험을 낮게 평가했던 진행자를 설득하여 실제로는 위험이 높으며 완화 조치가 필요하다는 점을 인정하게 되었습니다(2).

모든 구성원은 각 사용자 유형(관리 직원, 셀프 서비스 사용자 및 간호사)별로 별도의 API 게이트웨이를 구축하면 간호사 이외의 사용자가 의료 기록 인터페이스에 접근하는 것을 방지할 수 있다는 데 동의했습니다. 아키텍트가 최종적으로 설계한 아키텍처는 그림 22-12에 나와 있습니다.

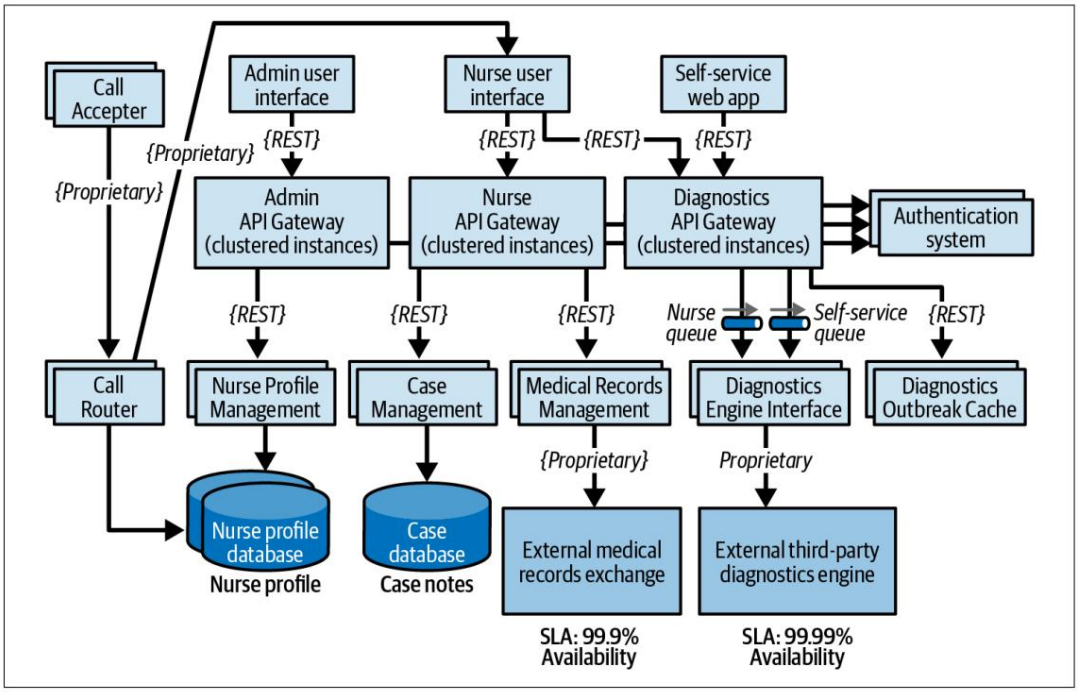


그림 22-12. 보안 위험을 해결하기 위한 아키텍처 수정

이 예시는 리스크 스토밍의 효과를 보여줍니다. 아키텍트, 개발자, 주요 이해관계자들이 협력하여 시스템 성공에 가장 중요한 아키텍처적 특성을 고려하고, 그렇지 않았다면 간과되었을 위험 영역을 식별합니다.

원래 아키텍처 (그림 22-8) 는 위험 폭풍 (그림 22-12) 이후 가용성, 탄력성 및 보안에 대한 우려를 해결하고 이 아키텍처를 더욱 효과적이고 성공 가능성이 높게 만드는 방식으로 크게 변경되었습니다 .

# 요약

리스크 스토밍은 일회성 프로세스가 아닙니다. 시스템 수명 주기 전반에 걸쳐 팀이 운영 환경에 나타나기 전에 위험 영역을 식별하고 완화하기 위해 지속적으로 노력하는 과정입니다.

조직에서 리스크 스토밍 세션을 얼마나 자주 개최해야 하는지는 변경 빈도, 아키텍처 리팩토링 작업, 아키텍처의 점진적 개발 등 여러 요인에 따라 달라집니다. 일반적으로 주요 기능을 추가한 후 또는 각 반복 작업이 끝날 때마다 특정 영역에 대한 리스크 스토밍을 진행하여 아키텍처가 올바르고 비즈니스 요구 사항을 충족하는지 확인합니다.