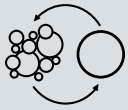
백서

오래된 애플리케이션을 새롭게 만드는 방법

지속적인 애플리케이션 현대화를 위한 소프트웨어 개발 패턴 및 프로세스

Zohaib Khan



"현재 우리가 수행하는 모든 작업은 미래의 레거시 소프트웨어를 작성하는 것입니다."

MARTIN FOWLER
THOUGHTWORKS 수석 아키텍트

핵심 요약

지난 수십 년 동안의 IT 및 소프트웨어 엔지니어링을 되돌아보았을 때 한 가지 분명한 사실은 모든 것이 변화한다는 점입니다. 하드웨어, 언어, 인프라, 방법론이 점진적으로 발전하던 시기는 패러다임 전환의 혁신으로 비로소 마무리되었습니다.

이러한 진화로 말미암아 IT 부문은 끊임없이 변화하는 비즈니스 요구에 부응할 수는 있었지만 그 과정은 쉽지 않았으며, 경제적이지도 않았습니다. 오래된 자산을 유지하는 데에는 많은 IT 예산이 소모되며, 최신 상태를 유지하기 위해 지속적인 업그레이드와 마이그레이션을 하는 것도 비즈니스 이점 실현 이전에 자금과 리소스를 고갈할 수 있습니다.

반면 적절한 접근 방식을 사용하면 저렴한 비용으로 신속하게 가치를 창출할 수 있는 애플리케이션 포트폴리오를 현대화할 수 있으므로, 제품 및 기술의 지속적인 진화에 맞춰 더욱 쉽고 경제적으로 최신 상태를 유지할 수 있습니다.

본 백서에서 Red Hat은 기존 애플리케이션을 현대화하는 데 사용되는 세 가지 특정한 소프트웨어 개발 패턴에 대해 살펴봅니다. 이러한 현대화 패턴은 기존 애플리케이션을 더욱 현대적인 아키텍처와 인프라로 전환하여 새로운 애플리케이션에 액세스가 가능하도록 지원합니다. 본 문서에서는 재작성(rewriting)이 유일한 옵션일 경우 재작성을 수행할 수 있는 조건에 대해서도 살펴봅니다. 이들 패턴은 기업이 기존 애플리케이션을 최대한 활용할 수 있는 방법을 파악하고, 현재와 미래의 비즈니스를 지원할 수 있는 지속적인 현대화 사례를 구축하는 데에 도움이 됩니다.

애플리케이션 개발 및 배포

얼마 전까지만 해도 애플리케이션은 프로그래밍 언어로 코딩되어 프로세서 및 운영 체제에 고유한 형식으로 컴파일되었습니다. 애플리케이션은 대체로 독립적(self-contained)이고 규모가 큰 경우가 많았으며 프라이빗 데이터센터에서 실행되었습니다. 그리고 모두가 애플리케이션의 수명이 길 것이라고 생각했습니다. 이러한 애플리케이션은 공식적, 사전적 요구 사항을 포함하면서 개발 기간이 긴 중량화된(heavyweight) 소프트웨어 개발 라이프사이클 접근 방식을 사용해 구축되었습니다.

하지만 이 모든 것이 바뀌었습니다. 이전의 애플리케이션은 이제 모놀리식(monolithic) 레거시 애플리케이션으로 불리며 시대에 한참 뒤떨어진 비즈니스 애플리케이션으로 간주됩니다. 이러한 애플리케이션은 애플리케이션 구축이라는 목적에는 부합했지만, 비즈니스의 속도 및 기술 혁신이 가속화됨에 따라 기업에게 부담이 되었습니다.

이후 혁신이 이루어짐에 따라 오늘날 일반적으로 사용되는 애플리케이션 개발 및 배포 모델이 출현했습니다. DevOps 프로세스는 클라우드에서 실행되는 컨테이너에 배포된 마이크로서비스의 생성을 처리합니다. 그리고 방법론, 아키텍처, 배포, 인프라의 네 가지 애플리케이션 개발 영역 각각에서 진행 상황만 고려하면 됩니다.

개발 프로세스의 진행 속도는 훨씬 빨라졌습니다. 사전적 요구 사항이 필요하고 사양에서 릴리스까지 오랜 시간이 걸리는 폭포수 방식(waterfall methodology)에서 릴리스가 빈번하게 수행되는 반복적 방법론(iterative methodology)으로 발전했습니다. 그리고 이제는 지속적이고 자동화된 통합 및 제공으로 고도의 협업이 이루어지는 DevOps 사례가 지원되고 있습니다.

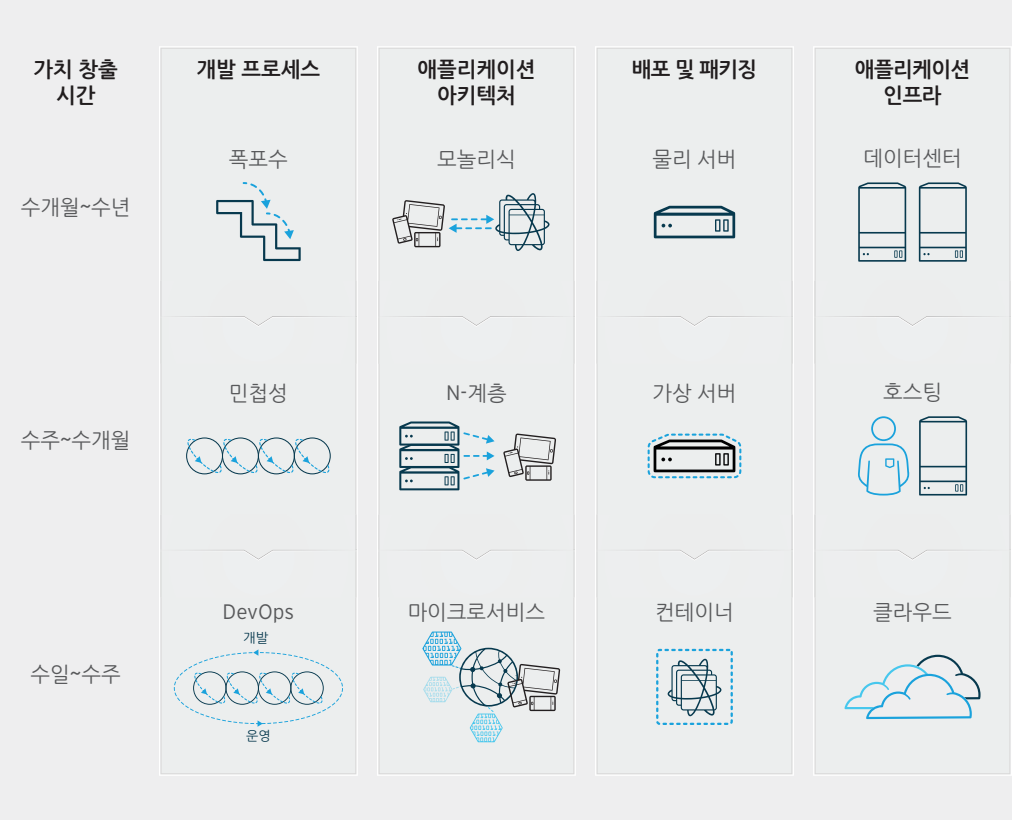


그림 1: 애플리케이션 개발 및 배포의 진화

전체 솔루션 스택에는 애플리케이션을 개발하고 배포하는 데 필요한 모든 구성 요소, 애플리케이션을 개발하는 데 사용된 방법론, 애플리케이션을 실행하는 하드웨어가 포함됩니다.

애플리케이션 아키텍처 부문에서는 기능을 여러가지 구성 요소로 분류하는 것이 트렌드가 되었습니다. 올인원 애플리케이션은 인터페이스, 로직, 데이터에 대해 각각의 티어로 분리되었습니다. 서비스 중심 아키텍처는 공통 엔터프라이즈 서비스 버스(ESB)를 통한 전용 서비스를 사용하여 애플리케이션을 구축할 수 있도록 해줍니다. 이러한 아키텍처는 다른 서비스 및 애플리케이션에서 고도로 전문화된 구성 요소에 액세스할 수 있는 마이크로서비스 및 API에 기반한 최신 모델까지 계속 이어졌습니다. 마이크로서비스 및 API는 구현(애플리케이션의 수행 방법 및 수행 기능)에서 인터페이스(애플리케이션에 액세스하는 방법)를 분리하기 위해 웹 서비스의 이전 표준을 기반으로 구축되며 다양한 언어로 구현할 수 있고 다른 시스템에 배포할 수 있습니다.

애플리케이션 배포도 훨씬 더 유연해졌습니다. 애플리케이션은 더 이상 하드웨어에 긴밀하게 연결되지 않습니다. 애플리케이션은 이제 Enterprise Java™ 등의 표준에 작성되어, 하드웨어와 운영 체제의 여러 조합에 배포할 수 있습니다. 또한 가상 머신과 컨테이너를 이용해 애플리케이션을 더욱 패키징하여 다른 여러 호스트에 간편하게 배포할 수 있습니다.

애플리케이션 인프라는 대규모의 애플리케이션 특정(application-specific) 서버에서 여러 애플리케이션을 지원하는 수평으로 확장된 상용 서버로 발전했습니다. 이제는 애플리케이션을 분산된 데이터센터, 프라이빗 클라우드, 퍼블릭 클라우드의 멀티플 서버에 배포하는 것이 일반적입니다. 이에 따라 배포 속도가 빨라지고, 성능 및 가용성도 향상됩니다.

이처럼 네 가지 영역 전반에 걸친 애플리케이션 개발 및 배포의 진화로 초기 개발을 더욱 신속하게 진행하고 업데이트 빈도를 늘리는 동시에 품질을 높이고 비즈니스 요구에 더욱 긴밀하게 연계할 수 있을 뿐만 아니라 운영상의 유연성을 대폭 개선하고 비용을 절감할 수 있게 되었습니다.

현대화의 핵심은 새로운 기술과 사례를 채택하는 것이 아니라 오래된 자산을 어떻게 관리하느냐에 달려 있습니다.

현대화

오늘날에는 애플리케이션을 지원하는 소프트웨어 구성 요소를 '스택'이라고 지칭합니다. 더 광범위한 관점에서 볼 때 전체 스택에는 애플리케이션을 개발하고 배포하는 데 필요한 모든 구성 요소, 애플리케이션을 개발하는 데 사용된 방법론, 애플리케이션을 실행하는 하드웨어가 포함됩니다.

기업들은 신규 프로젝트를 위해 새로운 제품과 기술을 수용했지만, 레거시 솔루션 스택은 여전히 많은 기업에서 사용되고 있습니다.

예를 들어 다수의 금융 서비스 기업들은 수십 년 전에 커스텀 애플리케이션을 개발하고, Bloomberg 터미널 및 거래자 워크스테이션, 클라이언트-서버 시스템, n-tier 웹 애플리케이션을 배포했습니다. 오늘날, 동일한 금융 기업들이 고객과 직원을 위한 모바일 애플리케이션을 제작하고 있습니다. 또한 성장과 도입의 결과로 애플리케이션 포트폴리오 전체에 걸쳐 수십 개의 현대화된 스택과 레거시 스택을 실행하고 있습니다.

현대화의 핵심은 새로운 기술과 사례를 채택하는 것이 아니라 오래된 자산을 어떻게 관리하느냐에 달려 있습니다. 예를 들어, 석탄을 이용해 난방을 하던 낡은 집의 일부분을 석유와 가스 난방이 되도록 수리한다고 생각해 보겠습니다. 만약 이 낡은 집의 전체를 태양열 난방으로 업그레이드한다면 비용이 많이 들고 당장 이윤을 얻을 수 없지만, 태양열 난방을 추가하는 것은 보다 합리적인 결정입니다.

애플리케이션 현대화에는 두 가지 주요 목표가 있습니다. 한 가지는 가능한 한 새로운 애플리케이션에서 기존 기능과 데이터를 사용(기존 애플리케이션에서 새로운 가치를 창출)하는 것이고 다른 하나는 새로운 프로세스, 제품, 기술의 이점을 오래된 애플리케이션에 가져오는 것입니다.

기업은 현대화에 적합한 접근 방식(적절한 방법론 및 패턴)을 통해 변화를 수용하고 앞서 나갈 수 있습니다.

현대화를 위한 세 가지 소프트웨어 개발 패턴

아래에서 설명하는 세 가지 소프트웨어 개발 패턴은 애플리케이션을 현대화하기 위한 통합 접근 방식을 제공합니다. 이 중에서 두 패턴은 기존 애플리케이션의 수명과 유틸리티를 확장하고 전체 포트폴리오가 한 번에 모두 변경되지 않도록 방지하여 애플리케이션 리팩토링 및 아키텍처 업데이트를 점진적으로 수행할 수 있도록 세 번째 패턴을 위한 장을 마련합니다. 어떤 경우에도 현대화로 이점을 얻기 위해 애플리케이션을 한꺼번에 다시 작성할 필요가 없습니다. 모놀리식 애플리케이션이라도 다시 작성하지 않고 현대화의 이점을 얻을 수 있습니다.

리프트 앤 시프트 (LIFT AND SHIFT)

리프트 앤 시프트는 기존 애플리케이션을 패키징하고 배포하는 방식을 현대화합니다. 리프트 앤 시프트를 이용해 기존 구성 요소를 현대화된 배포 플랫폼에 배포할 수 있습니다. 전형적인 예로 애플리케이션 가상화가 있는데, 애플리케이션이 운영 체제와 함께 패키징되고 전용 하드웨어 대신에 가상 머신으로 실행됩니다.

리프트 앤 시프트는 애플리케이션 아키텍처를 현대화하기 위한 것은 아닙니다. 그러나 이를 통해 기업은 현대화된 배포 플랫폼에서 애플리케이션을 실행할 수 있으며 나중에 애플리케이션을 리팩토링하기 위한 시간을 확보할 수 있습니다.

리프트 앤 시프트를 사용하여 더 빠른 최신 하드웨어에 배포함으로써 애플리케이션 성능을 개선할 수 있으며, 현대화된 플랫폼에서 이용 가능한 간단한 배포 프로세스로 애플리케이션의 유연성을 높일 수 있습니다. 또한 일회성 서버를 만료하고 관리를 중앙화하여 운영 비용을 절감할 수도 있습니다.

리프트 앤 시프트의 몇 가지 일반적인 예는 다음과 같습니다.

> 프레젠테이션, 비즈니스 로직, 데이터 서비스가 포함된 3 계층 구조(3-tier) 애플리케이션에는 다른 Linux® 서버에서 실행되는 각 계층이 있습니다. 각 서비스는 해당 서비스의 모든 설정 및 런타임 종속성을 포함하는 컨테이너로 다시 패키징됩니다. 컨테이너는 퍼블릭 클라우드에서 실행되는 PaaS(서비스로서의 플랫폼) 환경에 배포됩니다.
>
> J2EE에 구축되고 4개의 가상 머신에서 실행되는 CMS(컨텐츠 관리 시스템)는 컨테이너 세트로 다시 패키징되고 PaaS에 배포됩니다. 이로써 개발 팀은 PaaS에서 실행되는 컨테이너의 장점 외에도 지속적인 통합(CI)과 연속 배포(CD)를 포함하는 통합된 개발자 환경에서 이점을 얻을 수 있습니다.

위 예시를 살펴보면 아키텍처는 변경되지 않았지만 애플리케이션은 현대화된 배포 플랫폼에서 새로운 방식으로 작동하고 있습니다. 이를 통해 팀은 애플리케이션을 리팩토링하고 아키텍처를 현대화할 시간을 확보할 수 있습니다. 예를 들어 4개의 CMS 구성 요소는 마이크로서비스로 재설계할 수 있으며 전체 CMS는 API로 캡슐화할 수 있어, 모바일, 클라우드, 그리고 컨텐츠 관리 기능이 필요한 기타 애플리케이션에 액세스가 가능합니다.

리프트 앤 시프트가 모든 애플리케이션에 적합한 것은 아닙니다. 특정 벤더 솔루션에 종속된 애플리케이션은 다시 패키징하고 다시 배포하기가 어려울 수도 있습니다. 그리고 너무 오래된 운영 체제는 최신 배포 플랫폼에서 지원되지 않을 수도 있습니다.

새로운 레이어를 통한 보강

많은 기업이 모바일 및 Salesforce 등의 새로운 통로를 이용해 애플리케이션을 제공하고 파트너 애플리케이션과 통합하여 비즈니스 가치를 창출하고 있습니다. 새로운 레이어를 통한 보강은 기존 애플리케이션 기능이 새로운 애플리케이션 및 콘딧(Conduit)에 액세스하게 해 주는 소프트웨어 개발 패턴으로, 기능을 다시 개발할 필요가 없기 때문에 개발 시간과 비용을 줄일 수 있습니다. 또한 시간이 지나면서 자세히 검증되었으므로 기존 애플리케이션이 존재하는 곳에 복잡하고 중요한 기능을 사용하는 것이 좋습니다.

새로운 레이어를 통한 보강에서는 새로운 애플리케이션에 액세스가 가능한 인터페이스를 통해 기존 애플리케이션 기능 및 데이터를 래핑하는 애플리케이션 소프트웨어에 대한 새로운 레이어를 생성하는 작업이 수반됩니다. 과도한 복잡성을 방지하기 위해 새로운 레이어는 일반적으로 추가 비즈니스 로직을 포함하고 있지 않으며 단순하게 새로운 애플리케이션과 오래된 애플리케이션 간에 어댑터 역할을 합니다.

대부분의 경우 기존 애플리케이션을 수정할 필요가 없습니다. 이 패턴은 소스 코드를 사용할 수 없거나 기존 애플리케이션을 수정하는 것이 너무 위험할 때 유용하게 사용할 수 있습니다.

리프트 앤 시프트와 마찬가지로 기존 애플리케이션의 아키텍처는 변경되지 않지만 새로운 레이어를 통한 보강을 수행하면 기존 기능을 통해 현재 아키텍처를 이용하는 새로운 애플리케이션 및 서비스를 구축할 수 있습니다.

새로운 레이어를 통한 보강은 애플리케이션 아키텍처의 점진적인 현대화를 위한 장이 됩니다. 그리고 시간이 지나면서 오래된 애플리케이션 기능을 다시 작성하고 오래된 스택을 폐기할 수 있습니다. 그런 다음 한번에 모두 다시 작성하는 것이 아니라 이 패턴을 사용하여 기존 애플리케이션(Strangler 애플리케이션이라고도 함)을 서서히 마이그레이션할 수 있습니다.

다음은 새로운 레이어를 통한 보강의 몇 가지 예입니다.

> 기존 상용 애플리케이션은 현재 사용되고 있는 가상화 또는 컨테이너 플랫폼에서 지원되지 않는 운영 체제에서 실행되며, 표준 API를 통해 액세스할 수 있습니다. 해당 API에 액세스하는 어댑터 마이크로서비스가 작성됩니다. 그런 다음 이 마이크로서비스는 상용 애플리케이션의 기능에 액세스할 수 있도록 새로운 애플리케이션 및 다른 마이크로서비스에서 사용됩니다.
>
> Visual Basic으로 작성된 주문 애플리케이션은 해당 데이터베이스 백엔드의 복잡한 저장 프로시저를 광범위하게 활용합니다. 또한 휴대폰과 태블릿을 위한 새로운 모바일 주문 애플리케이션이 개발되고 있습니다. 어댑터 레이어는 새로운 애플리케이션에 대한 마이크로서비스 인터페이스를 통해 생성됩니다. 어댑터에는 데이터베이스 및 저장 프로시저가 표시되지 않습니다. 모든 새로운 애플리케이션은 요청을 저장 프로시저에 대한 호출로 변환하는 어댑터에 액세스합니다.

위의 두 가지 예시 모두에서 새로운 레이어를 통한 보강은 현재 아키텍처를 이용하는 새로운 개발에서 기존 애플리케이션을 사용할 수 있도록 해주는 임시 해결책일 뿐입니다. 이 패턴은 리프트 앤 시프트와 호환되며 함께 적용할 수 있습니다. 요약하면, 새로운 레이어를 통한 보강 패턴은 기존 애플리케이션의 기능을 새로운 애플리케이션에 사용할 수 있도록 해주고, 리프트 앤 시프트 패턴은 기존 애플리케이션을 더 간편하고 저렴하게 배포하고 관리할 수 있도록 해줍니다. 그리고 두 패턴 모두 개발자가 현재 아키텍처를 사용하여 기존 애플리케이션을 리팩토링하고 다시 작성할 수 있는 시간을 확보할 수 있게 해줍니다.

재작성 (REWRITE)

애플리케이션 재작성은 처음부터 새로운 애플리케이션을 만드는 것과는 다르며, 기존 애플리케이션을 대체하고 폐기하기 위한 새로운 기능을 만드는 프로세스입니다. 전반적인 현대화 전략의 일부로, 재작성은 리프트 앤 시프트와 새로운 레이어를 통한 보강 이후에 수행할 수 있으며 완전한 최신 스택을 갖추기 위해 애플리케이션 아키텍처를 업데이트할 수 있는 유일한 방법입니다.

일반적으로 기존 애플리케이션의 재작성은 세 가지 패턴 중 가장 유용하지 않은 옵션입니다. 왜냐하면 비용이 많이 들고 시간도 많이 소요되며 해당 비용을 상쇄하는 데 몇 년이 걸릴 수도 있기 때문입니다. 또한 애플리케이션이 새로운 비즈니스 가치를 제공하지 않는다면 예산이 제한된 상황에서 경영진에게 재작성의 명분을 내세우기란 어려운 일입니다.

그럼에도 불구하고 재기록이 유일한 옵션인 경우가 있습니다. 예를 들어 벤더에서 더 이상 지원하지 않는 운영 체제 및 하드웨어에서 매우 오래된 애플리케이션이 실행되고 있을 수 있습니다. 기업은 벤더가 지원하는 안전망 없이 중요한 비즈니스 시스템을 실행하기를 원치 않으며 오래된 시스템을 운영할 수 있는 기술을 보유한 인력이 전무한 경우도 간혹 있습니다.

재작성이 유일한 방법인 상황에서 최상의 접근 방식은 오래된 애플리케이션의 기능을 점진적으로 마이그레이션하는 것이며, 이것은 새로운 레이어를 통한 보강으로 가능합니다. 또한 일부 기능이 쓸모 없게 될 수 있고 전혀 마이그레이션할 필요가 없게 될 수도 있으므로 재작성 작업을 늦추는 것도 좋은 생각입니다. 오래된 행태를 단순히 마이그레이션하고 싶은 마음은 자제하고, 대신에 새로운 애플리케이션을 개발하는 것처럼 계획을 세우고 우선순위를 정하는 것이 좋습니다. 그러면 다가오는 변화에 대해 더 유연하고 융통성 있는 애플리케이션을 구축하는 데 도움이 될 수 있을 것입니다.

패턴 선택

최상의 단일 패턴은 없으며 이는 전적으로 애플리케이션, 비즈니스, 상황별 요인에 따라 달라집니다. 패턴은 애플리케이션 개발 및 배포의 다양한 영역을 다루기 때문에 멀티플 패턴은 하나의 애플리케이션에 적용할 수도 있고 이점이 전혀 없을 수도 있습니다. 일반적으로 포트폴리오를 현대화하는 데에는 두 가지 이상의 소프트웨어 개발 패턴이 필요합니다.

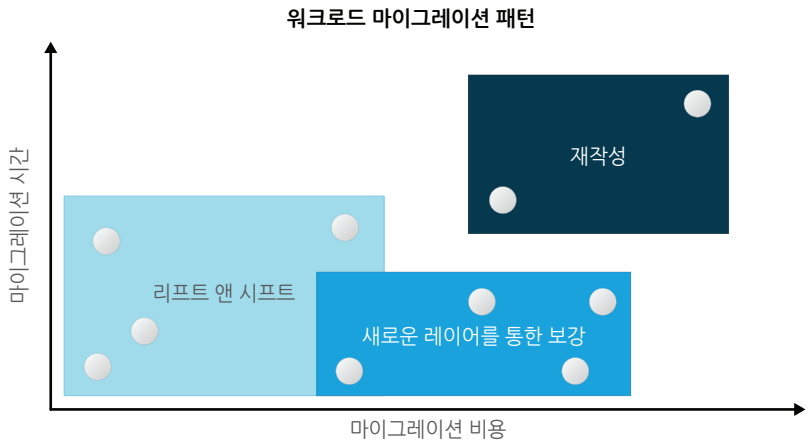


그림 2: 워크로드 마이그레이션 패턴은 비용 대비 시간에 따라 결정됩니다. 이를 통해 얻을 수 있는 이점은 애플리케이션과 애플리케이션의 현재 및 미래 상태에 따라 달라지므로 명시되지 않았습니다.

일반적으로 리프트 앤 시프트는 적용하는 데 가장 적은 비용이 드는 패턴입니다. 새로운 레이어를 통한 보강에는 개발 작업이 필요하므로 비용이 더 많이 들 수 있지만 대개는 더 신속하게 적용 가능합니다. 그리고 재작성은 대부분의 경우 비용이 가장 많이 들고 시간도 가장 많이 소요됩니다(그림 2).

이러한 소프트웨어 개발 패턴은 기업이 기존 애플리케이션을 최대한 활용하고 현재와 미래의 비즈니스를 지원할 수 있는 지속적인 현대화에 대한 사례를 구축하는 방법을 파악하는 데 도움이 됩니다.

현대화를 위한 방법론

세 가지 현대화 패턴은 애플리케이션 아키텍트와 개발/운영 관리자가 기존 애플리케이션에 새로운 활력을 불어넣을 수 있도록 도와 주지만, 대개는 더 규모가 큰 이니셔티브의 일부에 불과합니다. 이 경우 작업을 조직화하고 비용을 지속적으로 관리하고 비즈니스에 가치를 제공하는 데 공식적인 방법론이 도움이 됩니다.

Red Hat은 애플리케이션 현대화에 여러 단계의 반복 프로세스를 사용합니다(그림 3). 주요 목표는 계획을 세우고 이를 여러 번 실행하되, 각각의 반복에서 새롭고 점진적인 가치를 창출해 내는 것입니다. 이를 통해 리핑(ripping) 및 교체의 위험을 줄일 수 있습니다.

탐색 단계의 인터렉티브 세션을 통해 현재 애플리케이션 상태 및 목표를 파악합니다. 주요 이해관계자는 현대화하기 위한 잠재적인 방안을 모색하고 꾸준한 진행에 대한 약속을 받으려는 목적으로 단계(step)에 우선순위를 정하기 시작합니다.

그림 3: Red Hat의 애플리케이션 현대화 프로세스

설계 페이스(phase)에서 일련의 단계들은 분석, 기술검증, 파일럿을 통해 진행됩니다. 자동화된 애플리케이션 코드 분석은 잠재적인 문제를 파악하고 필요한 노력을 예측하는 데 도움이 되고, 사용자는 스택 및 아키텍처를 면밀하게 살핌으로써 현재 상태를 확인하고 현대화할 후보를 결정할 수 있습니다. 이 단계에서 원하는 상태가 설계되고 현대화 패턴이 선택되며, 상세한 배포 계획이 수립됩니다.

배포 계획은 다음 단계에서 실행됩니다. 위험을 줄이고 이전 단계에서 학습할 수 있도록 여러 번의 반복 실행으로 구성된 이 모델은 마이그레이션 전문가가 배치된 CoE(Center of Excellence)에서 지원해야 합니다. 이 그룹은 프로젝트의 진화 및 이해관계자의 변화에 따라 모범 사례를 캡처하고 지속 가능성을 보장하는 역할을 수행합니다.

결론

애플리케이션 현대화는 대형 애플리케이션 포트폴리오에 적용하는 경우, 특히 매우 오래된 스택 및 구식 하드웨어에서 애플리케이션을 실행하고 있는 경우에 힘든 작업이 될 수 있습니다. 현대화에 소요되는 시간과 노력을 예상하기란 어려운 일이며 운영 비용을 낮추는 것이 자금 조달을 정당화하기 위한 충분한 조건이 되는 경우는 흔하지 않습니다.

Red Hat의 방법론은 한 번에 몇 개의 애플리케이션만 마이그레이션할 수 있도록 문제를 분류합니다. 애플리케이션은 현대화, 사후 현대화 운영 비용, 잠재적인 비즈니스 가치에 소요되는 비용 및 시간을 기준으로 평가됩니다.

세 가지 현대화 패턴은 기존 애플리케이션의 수명을 연장하고 기존 애플리케이션을 처음부터 다시 작성해야 하는 시기를 결정하는 데 도움이 되는 몇 가지 표준 접근 방식을 제공합니다. 기업은 현대화에 맞는 접근 방식(적절한 방법론 및 패턴)을 통해 변화를 수용하고 앞서 나갈 수 있습니다.

Red Hat® Consulting은 앞서 설명한 패턴을 사용하여 조직이 애플리케이션에서 더 많은 가치를 얻을 수 있도록 지원하고 있습니다. 자세한 내용은 redhat.com/ko/resources/application-migration-modernization-consulting-jboss-middleware-datasheet를 참조하십시오.

저자 소개

Zohaib Khan은 현재 Red Hat에서 애플리케이션 마이그레이션 사례 책임자이자 PaaS CoP(Community of Practice) 관리자입니다. Zohaib는 Red Hat Summit , StackWorld, Red Hat Technical Exchange를 비롯한 다양한 컨퍼런스에서 발표했습니다. 그는 DevOps와 기술 변혁 및 혁신에 대한 글도 써오고 있습니다. Zohaib는 Red Hat에 합류하기 전에 의료 및 금융 서비스 부문 수석 관리자였으며 기술 서비스 회사를 공동 설립하여 기업가 정신을 시험하기도 했습니다.

한국레드햇 홈페이지 https://www.redhat.com/korea

RED HAT 소개

Red Hat은 세계적인 오픈소스 솔루션 공급업체로서 커뮤니티 기반의 접근 방식을 통해 신뢰도 높은 고성능 클라우드, Linux, 미들웨어, 스토리지, 가상화 기술을 제공합니다. 또한, 전세계 고객에게 높은 수준의 지원과 교육 및 컨설팅 서비스를 제공하여 권위있는 어워드를 다수 수상한 바 있습니다. Red Hat은 기업, 파트너, 오픈소스 커뮤니티로 구성된 글로벌 네트워크의 허브 역할을 하며 고객들이 IT의 미래를 준비하고 개발할 수 있도록 리소스를 공개하여 혁신적인 기술 발전에 기여하고 있습니다.

www.facebook.com/redhatkorea
구매문의 080-708-0880
buy-kr@redhat.com

www.redhat.com/ko
#046201_1216