

애자일(AGILE) 통합:

엔터프라이즈 아키텍처를 위한 청사진

저자: Steve Willmott, David Codelli

편집: Deon Ballard

비즈니스 성공에 있어 변화에 대응하는 능력이 점점 더 중요해지고 있습니다. 새로운 혁신적인 기업은 시장에 진입하고, 획기적인 기술의 등장으로 고객의 기대치가 뒤바뀜에 따라 조직은 그 어느 때보다 단축된 사이클로 비즈니스 계획을 변경해야 합니다. 현대적인 소프트웨어 아키텍처와 프로세스를 통해 조직은 이러한 변화를 효과적으로 처리하고 시장의 선두주자로 부상할 수 있습니다.

애자일(AGILE) 통합이라는 새로운 아키텍처 프레임워크는 컨테이너, 분산형 통합, API(애플리케이션 프로그래밍 인터페이스)라는 세 가지 중요한 아키텍처 기능을 통합합니다. 이 프레임워크는 이러한 핵심 기능을 통해 조직 내에서 민첩성을 향상하고 새로운 프로세스를 추진함으로써 경쟁 우위를 점할 수 있도록 합니다.

여행, 숙박 등의 산업이 새로운 비즈니스 방식을 통해 변화하였으며, 이제 새로운 서비스가 제공되어 소비자가 색다른 방법으로 서비스와 상호 작용하게 되었습니다. 이러한 혁신적인 변화의 동향은 신기술의 등장과 비즈니스-고객 간 상호 작용 방식에 대한 새로운 사고방식의 출현에 힘입어 금융 서비스 기업부터 공공기관에 이르기까지 주요 산업 전반으로 확산되고 있습니다. 또한 이러한 변화는 기존 조직에게 자사의 IT 기술을 근본적으로 혁신하여 새로운 서비스를 제공해야 한다는 압박감을 기존 조직에 안겨주고 있습니다.

최신 동향에 발맞춰 나가기 위해서 조직은 계획을 수립하여 소프트웨어 시스템을 신속히 변경할 수 있어야 합니다.

오늘날의 빠른 속도에 맞춰 소프트웨어를 제공하려면 조직에 애자일 인프라 기반이 필요합니다. 이 맥락에서 애자일은 애자일 소프트웨어 개발을 의미하지 않으며 전통적인 의미, 즉 유연하고 빠르게 움직일 수 있는 능력을 말합니다.

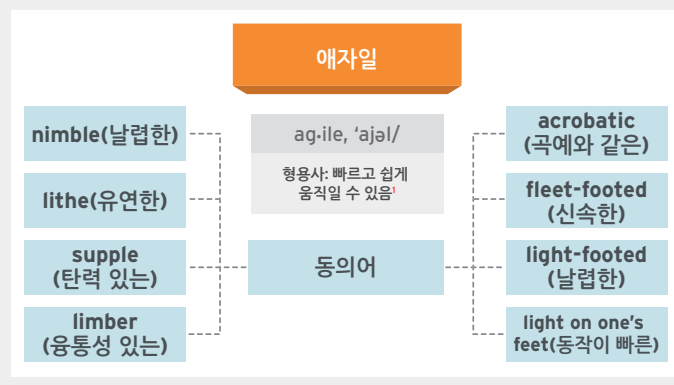


그림 1. 애자일 정의



지금까지 애자일 방법론은 소프트웨어 개발에 중점을 두어 애플리케이션 구축 방식을 개선하고 간소화하는데 주력했습니다. DevOps² 프랙티스를 통해서 이러한 방법론을 애플리케이션 배포 방식에 적용하고자 했습니다.

하지만 전체적인 관점에서 DevOps 자체의 변화를 살펴보면 조직에서 개발한 신규 소프트웨어 애플리케이션을 해결하는 수준에 도달했을 뿐입니다.

여기서 더 나아가 인프라 민첩성은 레거시 소프트웨어를 포함한 모든 IT 시스템을 포괄하는 환경을 구축할 수 있도록 합니다. 애자일 인프라는 기존 시스템, 다양한 데이터 유형, 데이터스트림, 고객 기대치라는 복잡성을 해결하고 이를 통합할 방법을 찾는 접근 방식입니다. 여기서 관건은 '통합'입니다.

3개월의 롤아웃 기간에 이어 수동 검증 단계를 차례대로 거쳐야 하는 조직에 비해 신속하게 가격을 변경하고 신제품을 출시할 수 있는 조직은 거대한 이점을 보유하고 있는 것입니다.

이를 애자일(AGILE) 통합이라 합니다. 통합은 인프라의 일부가 아니라 데이터, 애플리케이션, 하드웨어, 플랫폼이 포함된 인프라라는 개념의 접근 방식입니다. 통합 기술을 애자일 및 DevOps 기술에 맞게 조정함으로써 팀이 시장 수요에 따라 빠르게 변화할 수 있는 플랫폼을 구축할 수 있습니다.

계획의 무의미화: 조직과 민첩성

"다들 알고 계시듯 계획은 이제 의미가 없습니다." 2017 Red Hat Summit에서 Red Hat CEO인 Jim Whitehurst가 기조 연설에서 전달한 내용입니다. "잘 알려지지 않은 환경에서 세운 계획은 비효율적입니다."³ 비즈니스 환경이 가속화되고 변화로 인한 충동이 발생함에 따라 계획은 쉽게 중단되어 그 효과를 잃게 되며 작업 과정 중 어떤 한 단계에 갇히게 되면 매우 많은 비용이 발생하게 됩니다.

다시 말해, 보유한 정보가 적거나 환경이 덜 안정적인수록 계획의 가치가 저하됩니다.

무엇을 모르는지 스스로 깨닫는 것은 불가능

일반적으로 인프라 계획은 수 년에 걸친 장기적인 접근 방식을 취합니다. 다년간의 계획을 수립하려는 경우 시장 변화에 맞게 적절히 대처하거나 변화에 대한 결정을 조속히 채택하지 못할 수 있습니다. Jim Whitehurst가 언급한 '계획의 무의미화'는 결국 보다 빨리 계획을 수립하고 그 계획을 실행할 수 있는 역량을 의미합니다. 계획을 수립하고 새로운 계획을 만들기 위한 환경을 조성하는 데 걸리는 기간은 짧아야 합니다.

팀이 6개월 또는 24개월의 개발 사이클에 익숙해져 있다면 이러한 급격한 변화를 쉽게 받아들이기 어려울 수 있습니다. 전통적인 구조를 유지하고 있는 조직이 새로운 방법으로 시장에 진출하고 있는 스타트업 기업과 경쟁해야 하는 경우 이 문제는 더 심각해집니다. Netflix와 Blockbuster 또는 Uber와 전통적인 택시 서비스 등에서 이러한 현상을 목격할 수 있으며 1980년대의 개인용 컴퓨터나 1993년의 Amazon에서 시작하여 정보화 시대 초기에도 스타트업 기업의 혁신적인 효과가 이미 나타난 적이 있습니다.

² DevOps를 통한 혁신 가속화. <https://www.redhat.com/ko/insights/devops>

³ 2017 Red Hat Summit에서 Jim Whitehurst의 기조 연설. <https://www.cbronline.com/news/enterprise-it/software/red-hat-ceo-planning-know-dead/>

“민첩성을 기반으로 고객에게 지속적으로 서비스를 제공하기 위해서는 참여 시스템(SoE: System of Engagement)과 레코드 시스템(SoR: System of Record) 간의 인터페이스가 확장성뿐 아니라 적응성 측면에서도 민첩성을 구현해야 합니다. 예를 들어 기존 API에 새로운 속성을 추가하거나 향후 더 많은 컨텍스트를 제공할 수 있어야 합니다.”

HENRY PEYRET,
THE FORRESTER GROUP

Peyret, Henry. 'TechRadarTM: 통합 기술, 2015년 2분기' Forrester Research, Inc. 2015년 6월 23일.

표 1: 산업 전반에서의 혁신 기업(DISRUPTOR)

산업	전통적인 서비스	혁신적인 기업	영향
운송	택시, 대중 교통	Uber, Lyft	현지의 소규모 회사에서는 따라하기 힘든 일관된 고객 경험 창출
자산 관리	투자 회사	펀드 자동화	자금 관리에 있어 개인의 역량이 아니라 알고리즘이 차별화 요소가 됨
소매	오프라인 쇼핑	Amazon	쇼핑 방식이 오프라인 쇼핑에서 온라인 구매로 변화
검색 엔진	Google, 브라우저 기반 검색	음성 검색	Google의 주요 시장 채널에 영향을 미치고 새로운 기업이 시장에 진입할 수 있게 됨

스타트업 기업과 혁신적인 기업이 가진 장점은 인프라, 팀, 애플리케이션, 아키텍처 및 배포 프로세스까지 자유롭게 구성할 수 있다는 점입니다. Rachel Laycock이 '레거시 피플(legacy people)'⁴이라고 표현한 것 처럼 레거시 인프라에 발이 묶여 있지 않기 때문에 이들 기업은 혁신적인 아이디어를 생각해내는 것을 넘어서 아이디어를 실현할 수 있고 민첩하게 행동할 수 있습니다.

이들 조직은 새로운 것을 구축할 수 있는 능력이 있을 뿐 아니라 변화에 바로 적응할 수 있는 시스템도 구축할 수 있습니다. 소프트웨어 인프라는 이들이 다른 기업과 차별화되는 요소 중 하나이며, 변화하는 시장의 요구에 따라 시스템의 거의 모든 부분을 교체, 업데이트, 제거할 수 있습니다. 스타트업 기업 일부는 적응력이 아직 제대로 갖춰지지 않아 어려움을 겪기도 하지만 우수한 조직은 어떤 경우에도 기업 스스로 변화하는 능력을 유지합니다.

과제를 해결하는 역량

급변하는 환경에서 성공하려면 전체 IT 인프라가 애자일 방식으로 작동해야 합니다.

다음과 같이 두 가지 측면에서 변화가 일어나야 합니다.

- 아키텍처 설계에서 팀 커뮤니케이션에 이르기까지, 조직적이고 문화적인 차원에서 애자일 프로세스를 지원해야 합니다.
- 기술 인프라에서 기능을 신속하게 업그레이드, 추가, 제거할 수 있는 능력이 창출되어야 합니다.

기술적 변화와 문화적 변화가 민첩성을 실현하는 것이 아니라 민첩성을 위한 기반이 됩니다.

eBay의 제품 관리자인 Marty Cagan은 스스로 세금이라 칭하는 개념을 모든 프로젝트에 적용합니다. 즉, 매일 수행되는 일반적인 프로젝트에서 어느 정도의 시간과 리소스를 확보하여 신규 인프라 프로젝트에 투입하는 것입니다.⁵ 이렇게 함으로써 신규 프로젝트와 혁신을 최우선순위로 삼을 수 있습니다.

⁴ Rachel Laycock, '지속적 제공(Continuous Delivery)', 오후 세션, Red Hat Summit - DevNation 2016. 2016년 7월 1일, 캘리포니아주 샌프란시스코. <https://youtube.com/watch?v=y87SUSOfqTY>

⁵ Cagan, Marty, '인스파이어드(Inspired): 고객에게 사랑받는 제품을 만드는 방법(How to Create Products Customers Love)' Wiley Press, 2017

"새롭게 실험하는 것을 주저하다가 시장에서 경쟁업체에 밀리고, 민첩성을 실현하지 못하게 되면 그 기업은 침몰하게 됩니다. 기능을 도입하는 건 도박과 같습니다. 운이 좋다면 10%는 원하던 이점을 얻게 되죠. 즉, 새로운 기능을 시장에 더 빨리 출시하여 테스트할 수 있다면 더 큰 성과를 올릴 수 있습니다. 부수적으로는 투자를 빠르게 회수하여 자본 활용도를 높일 수 있으므로 수익 창출 속도도 빨라집니다."

GENE KIM,
THE PHOENIX PROJECT

Gene Kim, Kevin Behr, George Spafford, *Phoenix 프로젝트: IT, DevOps, 비즈니스 성공에 관한 이야기*. 오리건주 포틀랜드: IT Revolution Press, 2013.

인프라의 민첩성

여러 그룹이 개선을 위한 방법을 모색하기 위해 서로 다른 방향으로 움직이고 있기 때문에 신기술을 대량으로 도입하는 것이 애자일 인프라를 구축하는 데 도움이 되지 않는 경우가 많습니다. 최상위 수준의 목표를 일관성 있게 수립하지 못하면 조직의 전체 운영에 진정한 변화를 가져오기 위해 어떤 새로운 기능을 도입해야 할지 결정하기 힘들어집니다.

애자일 통합의 세 가지 핵심 요소

세 가지 주요 기술이 애자일 통합 접근 방식을 뒷받침합니다.



- 1. 분산형 통합:** 수십 개의 고급 통합 패턴이 엔터프라이즈의 작업과 데이터 흐름을 반영합니다. 이러한 통합 패턴이 컨테이너 내에 배포되어 있으면 특정 애플리케이션 및 팀의 요구에 부합하는 위치와 규모에 맞게 통합 패턴을 배포할 수 있습니다. 이는 전통적인 중앙 집중식 통합 아키텍처가 아닌 분산형 통합 아키텍처이며, 이 아키텍처를 기반으로 개별 팀이 적절한 통합 패턴을 민첩하게 정의하여 배포할 수 있습니다.
- 2. API:** 안정적이고 잘 관리된 API는 팀 간의 협업, 개발, 운영에 큰 영향을 미칩니다. API는 안정적이고 재사용 가능한 인터페이스에서 주요 자산을 래핑하여 인터페이스가 조직 전반에서, 파트너와의 관계에서 그리고 고객과의 관계에서 재사용할 수 있는 빌딩 블록으로 작동하게 만듭니다. API는 컨테이너와 함께 다양한 환경에 배포할 수 있으며 이를 통해 여러 사용자가 서로 다른 API 세트와 상호 작용할 수 있습니다.
- 3. 컨테이너:** API와 분산형 통합 기술 모두에서 컨테이너는 기본적인 배포 플랫폼 역할을 합니다. 컨테이너를 사용하면 특정 환경 내에 적절한 서비스를 배포하여 일관적이고 쉬운 방법으로 개발, 테스트, 유지관리를 수행할 수 있습니다. 컨테이너는 DevOps 환경과 마이크로서비스에서 가장 많이 사용되는 플랫폼이기 때문에 컨테이너를 통합 플랫폼으로 사용하면 개발팀과 인프라팀 간에 훨씬 더 투명하고 협력적인 관계를 구축할 수 있습니다.

이 세 가지 기술은 각각 고도의 추상화 수준을 구현하여 서로 다른 팀이 협업할 수 있게 하므로 IT 인프라가 더 민첩해 집니다. API 및 분산형 통합 기능과 함께 컨테이너 플랫폼을 사용하면 통합 자체에서 통합 구현을 추상화할 수 있습니다. API 및 분산형 통합 패턴은 기본 인프라를 파악하거나 변경할 필요 없이 광범위하게 이해가 가능한 수준에서 특정 자산을 패키징화하기 때문에 팀의 민첩성을 향상시킬 수 있습니다.

이러한 기술은 각각 특정 통합 과제에 대해 뛰어난 민첩성을 제공하며 함께 사용하면 그 효과는 몇 배가 됩니다. 기술을 뒷받침하는 것은 문화입니다. DevOps 프랙티스, 특히 자동화와 배포 프로세스를 결합하면 기술의 이점이 더 강력해 집니다.

분산형 통합

현재 IT 시스템이 직면한 최대 과제는 조직 전반에 애플리케이션을 연결해야 한다는 것입니다. 통합 서비스에 수반되는 문제는 더 복잡하고 중앙 집중화된 통합 허브라는 결과를 낳았습니다. 이러한 허브는 ESB(Enterprise Service Bus)로 구현되어 매우 복잡한 병목 현상을 일으켰으며 신속한 변화에 대비하기 위한 융통성이 결여되어 있습니다.

분산형 통합을 사용하면 이전 세대의 ESB가 지향했던 기술적 목표를 조직 내의 여러 팀에게 더 익숙한 방식으로 달성할 수 있습니다. ESB와 마찬가지로 분산형 통합 기술은 트랜스포메이션, 라우팅, 파싱, 오류 처리, 경고 기능을 제공하지만 차이점은 통합 아키텍처입니다.

분산형 통합 아키텍처는 각 통합 지점을 대규모 중앙 집중식 통합 애플리케이션의 일부가 아닌 별도의 고유한 배포로 처리합니다. 따라서 통합은 조직 전체에 배포된 다른 통합에 영향을 미치지 않고 특정 프로젝트나 팀을 위해 로컬로 컨테이너화되고 배포될 수 있습니다. 이러한 분산형 접근 방식은 애자일 프로젝트에 필요한 유연성을 제공합니다. 또한 기본적인 컨테이너 플랫폼에서 애자일 또는 DevOps 팀과 동일한 틀체인을 사용함으로써 팀이 자체 틀과 일정을 통해 통합을 더 효과적으로 관리할 수 있도록 합니다. 여기서 통합은 마이크로서비스로 다뤄지기 때문에⁶ 개발 속도와 릴리스 통합의 속도가 향상됩니다.

개발자 틀과 프로세스에 맞게 조정하는 것이 중요합니다. 분산형 통합의 핵심은 한 부서에서 여러 전문 사용자가 개발하고 관리하는 중앙 집중식 소프트웨어 인프라가 아니며 소프트웨어 개발 프로세스와는 별개로 배포된다는 것입니다. 공통 플랫폼 및 툴링으로 통합 아키텍처를 배포하면 프로젝트 수준에서 모든 개발자가 액세스할 수 있고 통합이 필요할 때 마다 경량의 배포를 지원할 수 있습니다.

표 2. 소프트웨어 라이프 사이클의 각 단계에 대한 통합 기술 비교

라이프 사이클 단계	ESB, 대부분의 IPAAS(INTEGRATION PLATFORM AS A SERVICE: 서비스로서의 연동 플랫폼)	분산형 통합 기술 지원
버전 관리	독점	GitHub 및 기타
빌드	독점	Maven 및 기타
배포	독점	컨테이너 및 기타 DevOps 툴
관리 및 확장	독점	컨테이너 및 기타 DevOps 툴

⁶ Martin Fowler의 유용한 마이크로서비스 정의 참조: <https://martinfowler.com/articles/microservices.html>

"소프트웨어에서 문제가 발생했을 때는 사용 횟수를 줄이는 것이 아니라 늘리는 것이 문제를 완화하는 방법입니다."

DAVID FARLEY,

지속적 제공: 빌드, 테스트, 배포 자동화를 통한 안정적인 소프트웨어 릴리스

David Farley와 Jez Humble, 지속적 제공: 빌드, 테스트, 배포 자동화를 통한 안정적인 소프트웨어 릴리스. Addison-Wesley Professional, 2010.

ESB를 사용하려면 팀은 개발 및 운영 환경에서 사용하고 있는 툴 외에 전체 라이프 사이클에서 ESB 툴을 사용해야 합니다. 이러한 제한으로 인해 운영에서 효율성이 떨어지고 여러 가지 문제와 오류가 발생하기 쉬워집니다.

메시징을 통해 통합 강화

분산형 통합은 그 구조상 통합을 마이크로서비스로 다루며 컨테이너화가 가능하고 로컬에 쉽게 배포할 수 있을 뿐 아니라 릴리스 사이클도 매우 짧습니다.

통합 기술은 이러한 경량의 마이크로서비스 기반 아키텍처를 지원할 수 있어야 합니다. Red Hat® JBoss® Fuse를 사용하면 사용자가 통합을 코드로 처리할 수 있으며 컨테이너를 포함한 모든 곳에서 이를 실행할 수 있습니다.

또한 JBoss Fuse는 Red Hat JBoss AMQ와 번들링되어 메시징 인프라를 제공합니다. 강력한 메시징 인프라를 통해 시스템 간에 이벤트와 데이터가 효과적으로 라우팅될 수 있습니다. 메시징은 비동기식 특성을 가지고 있어 종속성을 필요로 하지 않기 때문에 마이크로서비스 기반의 중요한 구조적 톨이 됩니다.

통합과 메시징을 결합하여 더 효과적인 라우팅, 여러 언어 및 프로토콜 지원, 비동기적 처리량, 데이터 관리 향상을 통해 통합 아키텍처의 전반적인 성능을 향상시킬 수 있습니다.

동향에 발맞추기

컨테이너 도입률은 점점 높아지고 있습니다. 그렇다면 어느 정도로 높아지고 있을까요? 그리고 그 이유는 무엇일까요? 451 Research에 따르면 컨테이너는 시장에서 250% 성장할 것이라 예측합니다.⁷ 단, 이는 컨테이너 배포가 아닌 사용에 해당하는 수치입니다. 실제 컨테이너 배포는 측정하기가 더 어렵습니다. Red Hat의 의뢰로 실시된 Bain 설문조사 결과에 따르면, 현재 약 20%의 고객이 프로덕션 환경에서 컨테이너를 배포하고 있으며 개발 및 테스트 환경에서도 거의 동일한 비율로 배포하고 있지만, 30% 이상의 고객은 컨테이너를 평가하고 있거나 기술검증(POC)을 실행하고 있는 것으로 나타났습니다.⁸

컨테이너를 사용한다는 것이 어떤 의미인지에 관해 명확하지가 않은 상황입니다. 기업 프로젝트에서 컨테이너를 도입하는 패턴은 대략 네 가지입니다. 일반적인 개발 또는 배포 플랫폼으로 사용하거나, 클라우드 네이티브 또는 마이크로서비스 플랫폼으로 사용하거나, 하이브리드 클라우드 내에서 사용하거나, 혁신적인 프로젝트를 위해 사용하는 것입니다.⁹ 컨테이너를 어떻게 사용하고 있는가에 따라 컨테이너 채택 방식이 달라질 수 있습니다.

애자일 통합에서는 기존 운영을 지원할 수 있는 인프라 플랫폼을 구축하는 것이 핵심입니다. 모든 구현 패턴에서 플랫폼을 차용할 수 있으나 그 핵심을 보면 신규 프로젝트와 기존 서비스 모두를 위한 기반이 되는 플랫폼으로 작동합니다.

7 클라우드 지원 기술 모니터 리포트에 따른 451 Research 인포그래픽, 2017년 1월. https://451research.com/images/Marketing/press_releases/Application-container-market-will-reach-2-7bn-in-2020_final_graphic.pdf

8 Bain 설문조사: 전통적인 기업에서 디지털 트랜스포메이션을 실현하는 방법과 컨테이너의 역할, 2016년 11월. <https://www.redhat.com/ko/resources/path-digital-containers>

9 <https://enterpriseproject.com/article/2017/8/4-container-adoption-patterns-what-you-need-know>

**“흔히 디지털
트랜스포메이션이라
불리는 새로운 경쟁의
장에서 기업은 IT
아키텍처를 재고하고
온프레미스 인프라와
클라우드 전반에서
워크로드를 재분산하며
상호운용을 통해
비즈니스 전략과 운영을
지원할 수 있어야 합니다.
이 모든 변화를 위해서는
통합에 대한 새로운 접근
방식, 즉 '하이브리드
통합'이 필요합니다.”**

CARL LEHMANN,
451 GROUP

Carl Lehman, 451 Research,
'새로운 하이브리드 통합 플랫폼
시장에서 iPaaS와 API의 혁신적인
역할'. 2017년 7월.
[https://451research.com/
report-long?icid=3862](https://451research.com/report-long?icid=3862).

컨테이너

가상화, 클라우드, 컨테이너는 모두 유사한 기술로서 그 목표도 비슷합니다. 이러한 기술은 물리 서버와 별개로 소프트웨어 운영 환경을 추상화하여 하드웨어에 더 많은 인스턴스를 스택할 수 있도록 하고 가용성, 확장성을 제공하며 효과적인 배포가 가능하게 하지만 서로 다른 방식으로 과제를 해결합니다. 가상화는 운영 시스템 레이어를 추상화합니다. 클라우드는 영구적이며, 전용 서버 인스턴스라는 개념을 사용하지 않습니다. 컨테이너는 단일 애플리케이션을 실행하기 위한 라이브러리와 운영 체제의 적절한 버전을 정의합니다.

컨테이너 기술에 기반을 둔 경량의 규범적인 접근 방식을 통해 컨테이너는 현대적인 소프트웨어 환경에 이상적인 툴이 되었습니다. 각 인스턴스는 운영 체제에서 컨테이너에 포함된 각 라이브러리의 정확한 버전에 이르기까지 불변의 정의를 사용합니다. 이로써 각 인스턴스에 대해 환경을 반복적으로 일관성있게 유지하므로 지속적 통합과 지속적 제공(CI/CD) 파이프라인에 적합합니다. 이뿐만 아니라 컨테이너 이미지는 단일 애플리케이션에 필요한 요소만 정의하기 때문에 컨테이너가 마이크로서비스와 일치하고 컨테이너 오케스트레이션은 대규모 마이크로서비스 인프라의 배포와 관리를 오케스트레이션할 수 있습니다.

경량성과 반복성의 결합은 컨테이너가 애자일 통합에 적합한 기술 플랫폼이 되도록 합니다.

전통적인 통합 접근 방식에서는 ESB가 인프라의 주요 지점에 배치된 고도로 중앙 집중화된 구조가 사용되었습니다. 분산형 통합 및 API 관리에서는 분산된 아키텍처를 통해 특정 위치 또는 팀에 필요한 기능을 배포할 수 있습니다. 컨테이너는 불변의 특성 때문에 전체 환경에서 이미지와 배포가 일관되게 유지되어 두 접근 방식을 위한 기본 플랫폼으로 동작하므로 불투명한 종속성이나 충돌 없이 빠르게 배포하거나 교체할 수 있습니다.

통합을 사용하든 API를 사용하든 배포 아키텍처의 핵심은 복잡한 승인 프로세스 없이 새로운 서비스를 설계하고 배포할 방법이 있어야 한다는 것입니다.

컨테이너를 사용하면 분산형 통합과 API를 마이크로서비스로 다룰 수 있습니다. 개발팀과 운영팀 모두를 위한 공통된 툴링을 제공하고 관리형 릴리스 프로세스와 함께 신속한 개발 프로세스를 사용할 수 있습니다.

컨테이너에는 오케스트레이션 필요

마이크로서비스가 별도의 단일 기능을 나타내는 것처럼 각 컨테이너는 단일 서비스 또는 애플리케이션을 나타냅니다. 마이크로서비스 아키텍처에는 수십 개에서 수백 개에 이르는 개별 서비스가 있을 수 있으며 이는 개발, 테스트, 운영 환경 전반에 걸쳐 복제됩니다.

이렇게 많은 수의 인스턴스가 있기 때문에 인스턴스를 오케스트레이션하고 고도화된 관리 태스크를 수행하는 능력은 효과적인 컨테이너 환경을 위해 반드시 필요합니다.

Red Hat OpenShift는 Docker 컨테이너를 Google의 Kubernetes 오케스트레이션 프로젝트와 결합하며 인스턴스 관리, 모니터링, 로깅, 트래픽 관리, 자동화 등의 중앙 집중식 관리를 포함합니다. 이는 컨테이너만 있는 환경에서는 거의 제공하기 어려운 기능입니다.

또한 Red Hat OpenShift는 셀프 서비스 카탈로그, 인스턴스 클러스터링, 애플리케이션 지속성, 프로젝트 수준 격리와 같이 개발자에게 친숙한 툴을 제공합니다.

이러한 결합을 통해 특히 안정성과 테스트라는 운영 요구 사항 그리고 간편한 사용과 신속한 제공이라는 개발자 요구 사항 간에 균형을 이룰 수 있습니다.

API

대부분의 정보 인프라에는 수백 또는 수천 개의 시스템, 애플리케이션, 자산이 포함되어 있지만 시스템에서 상호 작용이 이루어지기는 쉽지 않으며 어떤 시스템이 사용 가능한지 IT 관리자가 파악하는 것도 불가능할 수 있습니다.

API는 통합 기술을 사용하여 연결되는 모든 자산에 대한 인터페이스입니다. **API**는 애플리케이션 간 통신 방법을 설정하는 정의 또는 룰 세트입니다.

조직이 중앙 집중식의 통합 기술 센터에 기반한 접근 방식으로부터 분산된 접근 방식의 셀프 서비스로 이동하는 것이 주요 우선순위가 되었습니다. 애자일 팀에는 회사 내부와 외부에서 개발된 서비스를 찾아내고, 테스트하고, 사용할 수 있는 권한과 자율성이 부여되어야 합니다. 강력한 **API** 기능은 이러한 권한과 자율성을 팀에 제공합니다. **API**를 사용하면 팀에서는 원하는 수준의 통합을 실현할 수 있으며 조직에서는 보안, 권한 부여, 사용 정책을 관리 및 시행할 수 있습니다. **API**는 팀에 통합을 어떻게 설계할지 결정할 때 참고 자료가 되어 줍니다.

API는 최종 애플리케이션과는 다르며 애플리케이션이 어떻게 상호 작용하는지를 결정한 다음 개별 개발자가 프로젝트 내에서 이를 빌딩 블록으로 사용합니다. **API**는 개발자와 팀이 함께 사용할 수 있는 공통의 언어를 제공합니다. 조직은 **API**를 사용하여 커뮤니티를 지원함으로써 공유와 협력을 통해 새롭고 혁신적인 서비스 사용 방식을 개발할 수 있도록 합니다.

다양한 **API** 또는 서로 다른 **API**의 서브셋을 여러 대상에게 제공할 수 있습니다. 벤더의 요구 사항은 내부 개발팀이나 커뮤니티 개발자의 요구 사항과는 다를 수 있습니다. **API** 관리에는 애플리케이션과 사용자 그룹을 위한 **API**를 설계하는 것뿐 아니라 **API** 라이프 사이클을 관리하는 것도 포함됩니다. **API**는 점점 더 제품처럼 관리되고 있고 여러 팀이 각 **API**를 담당하고 있지만 모든 리소스 전반에서 일관성과 편의성을 보장해야 할 필요가 있습니다.

분산형 통합과 마찬가지로 컨테이너를 **API**를 개발, 배포, 관리하기 위한 하나의 플랫폼처럼 사용할 수 있으며 대규모 개발 및 운영 프로세스와 툴에 맞춰 **API**를 개발할 수 있습니다.

적합한 API 플랫폼을 통해 개발자 효율성 증대

API의 강점은 내부 개발자와 외부 사용자가 모두 사용할 수 있다는 데 있습니다. Red Hat 3scale API Management Platform은 모든 사용자를 위한 툴을 제공합니다. 또한 개발자에게는 개발자 포털을 제공하여 협업을 통해 API를 개발할 수 있고 관리 포털을 통해 이러한 API를 게시할 수 있습니다.

3scale API Management Platform은 인증을 제공하고 주요 클라우드 제공업체와 통합하며 컨테이너 내에서 실행하여 외부에서 API를 사용할 수 있게 합니다.

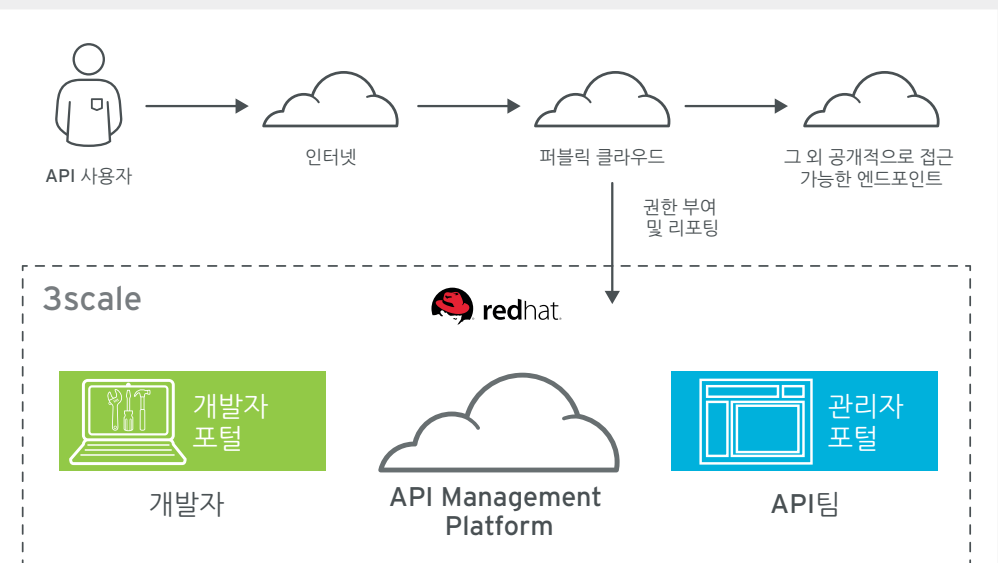


그림 3. API 관리, 엔드포인트, 퍼블릭 클라우드

API 전략에는 퍼블릭 API를 위한 API 설계가 결합되어 있습니다. 3scale API Management Platform, 특히 컨테이너 플랫폼상에 구축된 3scale은 이러한 전략을 이행하기 위한 수단을 제공합니다.

애자일 통합의 아키텍처

팀 프랙티스

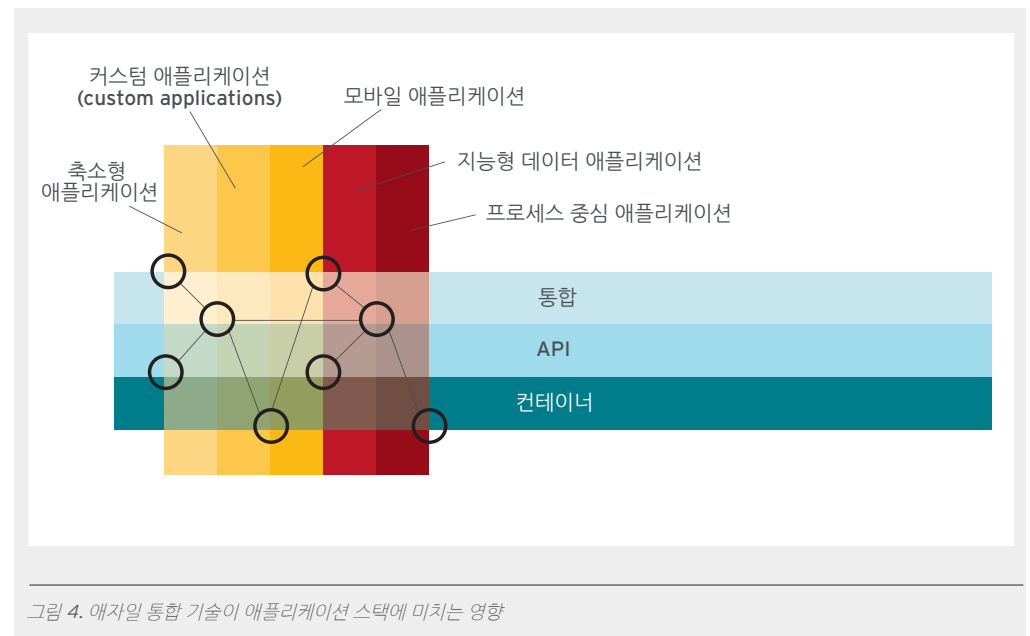
애자일 통합의 핵심 기술은 재사용 가능한 기능으로 팀에 배치 및 제공될 때 가장 큰 효과를 발휘합니다.

여기서 '기능'이란 허가된 그룹이 셀프 서비스 방식으로 기술을 사용하고, 조직의 지침을 쉽게 따르며, 모범 사례 정보에 액세스할 권한을 얻는 것을 말합니다. 정보 아키텍트나 IT 관리자는 개별 팀을 위해 다음과 같이 명확한 프로세스를 정의해야 합니다.

- 광범위하게 사용 가능한 사용 지침 제공
- 적절한 경우 사용 방식과 모범 사례를 적용하되 해당 규칙을 넘어 실험을 할 수 있는 자유를 허용
- 프로토타입에서 테스트, 실행, 업데이트, 사용 만료 단계까지 이동할 수 있도록 명확히 정의된 프로세스 확립
- 신규 배포 및 개발을 위한 정보 공유 허용
- 인프라팀이 모든 프로세스에 관여하게 만드는 것이 아니라 셀프 서비스 기능을 구축하고 제공하는 역할을 하도록 지원

예를 들어 소프트웨어팀이 전체 셀프 서비스 방식으로 새 API를 실행하기 위해 개발, 테스트, 준비하는 것이 가능하며 다른 그룹과 문서를 업데이트하기 위한 프로세스를 갖출 수 있습니다. 게시 또는 프로덕션 단계로 넘어가기 전에 다른 팀과 함께 진행해야 할 프로세스나 크로스 체크가 있을 수 있으나, 인프라를 통해 이러한 프로세스를 최대한 자동화할 수 있습니다.

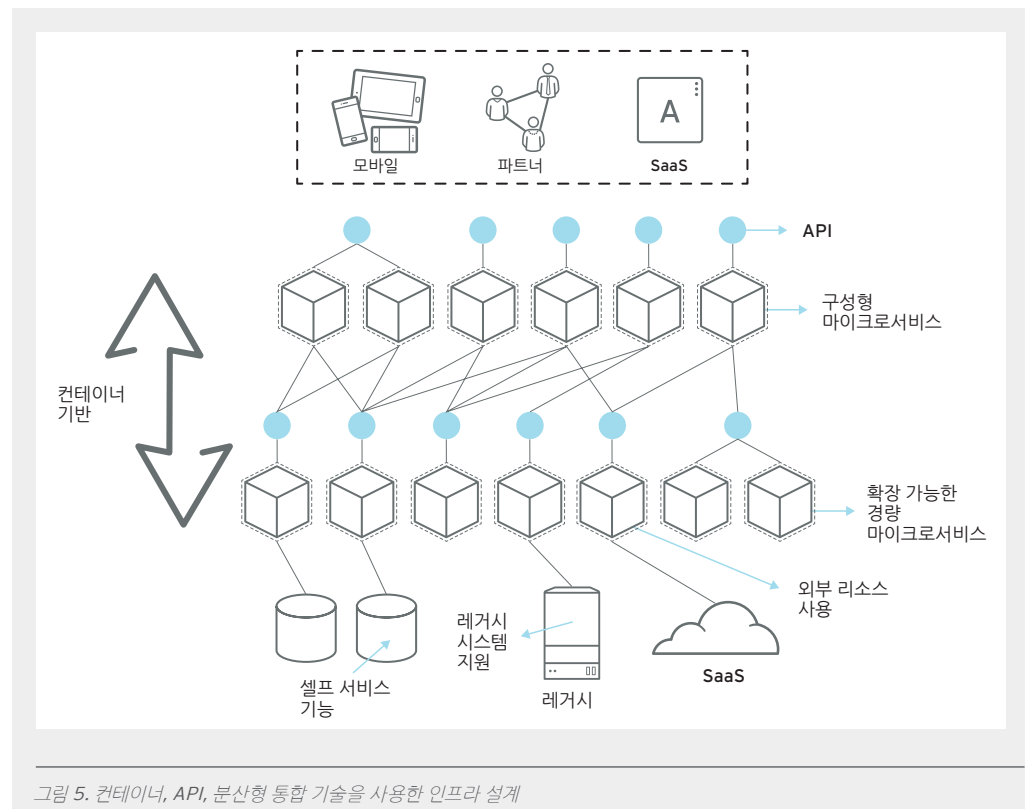
인프라 아키텍처



컨테이너, API, 통합을 동시에 활용하면 조직 내부의 소프트웨어 에코시스템에 대한 견고한 기반 계층이 구축되며 대부분의 경우 외부 통합을 위한 액세스 지점도 생성됩니다.

다양한 시스템에서 재사용 가능한 여러 엔드포인트를 제공합니다. 이러한 엔드포인트 각각은 재사용 가능한 API로 표시되며 컨테이너 내에서 실행되어 확장성과 배포 편의성을 구현합니다. 통합 프로세스에서는 개별 서비스 그룹을 통합하거나 조직의 여러 부분에서 얻은 결과를 수집하여 시스템 전반에서 필요한 곳에 트랜스포메이션, 컴포지션 또는 인라인 비즈니스 로직을 제공합니다.

통합된 애플리케이션은 최종 사용자 애플리케이션을 제공하기 전에 더 긴밀하게 연결될 수 있습니다.



모든 시스템이 점점 더 작은 부분으로 분해되고 여러 레이어의 API 추상화를 통과한다는 가정은 없습니다. 이러한 운영은 효율성을 낮추고 지연 시간을 늘리며 자체적으로 불필요한 복잡성을 가중시킬 수 있습니다. 일부 영역에서는 기존의 레거시 ESB 기능을 유지하면서 특정 애플리케이션 간의 연결을 그대로 두는 것이 적절한 선택일 수 있습니다. 분산 시스템 간의 종속성 또한 적절한 툴을 사용하여 추적하고 관리해야 합니다.

하지만 시스템 전체를 고려했을 때 컨테이너, API, 통합 측면에서 아키텍처를 재구성하는 것은 서비스, 통합 지점, 고객 상호 작용에 대해 적절한 선택을 할 수 있다는 의미가 됩니다. 예를 들어 대량의 인바운드 요청이 들어오면 보안을 점검한 후 올바른 백엔드 서비스에 바로 라우팅할 수 있으며 이때 ESB 병목 현상은 발생하지 않습니다.

분산 하이브리드 클라우드 환경에는 문제의 백엔드 시스템이 다양한 물리적 위치에 다수 존재할 수 있습니다. 로컬 요구를 충족하기 위해 인접한 로컬 시스템을 통합하면 주요 비즈니스 로직이 포함된 단일 중앙 통합 시스템을 통해 모든 것을 라우팅하는 것보다 높은 효율성과 보안을 실현할 수 있습니다.

애자일 조직과 문화

인프라의 라이프 사이클은 소프트웨어 개발이나 운영의 라이프 사이클과는 매우 다릅니다. 개발 사이클에서는 하나의 프로젝트를 완료한 후 다음 프로젝트로 이동하게 되며, 제품을 얼마나 빨리 출시하느냐 또는 주어진 시간에 기능을 얼마나 많이 생성하느냐가 효율성의 기준이 됩니다. 유지관리와 안정성이 중시되는 운영 사이클에서도 보안 패치와 업데이트를 적용하거나 새로운 서비스를 배포하거나 변경 사항을 보다 신속하고 효율적으로 롤백하는 것이 여전히 큰 이점을 가져다줍니다.

하지만 인프라에는 상당히 다른 접근 방식이 있습니다. 인프라는 고도로 전문화된 여러 그룹이 많은 시간을 들여 구축하는 것이 대부분이며 다기능팀(Cross-Functional Team)이 특정 소프트웨어 엔지니어링 프로젝트를 진행하는 것과는 매우 다릅니다. 인프라 프로젝트는 일반적으로 소프트웨어 프로젝트보다 규모가 훨씬 크며 릴리스 사이클이 짧은 경우 많은 성과를 내는 것이 불가능해질 수 있고 불완전한 상태로 남겨질 수도 있습니다. 엔터프라이즈 IT 전문가인 **Andrew Froelich**는 **InformationWeek**에서 인프라가 특히 하드웨어와 데이터센터와 관련하여 귀환 불능 지점이라는 제약을 가진다고 밝힌 바 있습니다. 퍼블릭 클라우드의 경우에도 더 이상 프로젝트를 폐기하고 다시 시작할 수 없는 지점이 존재합니다.¹⁰ 인프라는 영구적입니다. 하지만 인프라 성능에 관해 방법론을 조정할 수 있습니다.

애자일 및 DevOps와 같은 응답성 높고 반복적인 프로세스를 통해 개발팀과 운영팀은 큰 이점을 얻을 수 있는 반면 인프라팀은 그렇지 않습니다. 하지만 **Froelich**의 애자일 인프라 찬반 양론 분석에서는 한 가지 중요한 측면이 빠져 있습니다. 바로 인프라팀을 개발팀과 운영팀에 연계하는 일입니다. **Rohan Pearce**는 CIO에 인프라팀을 기능팀이 아니라 애자일 스타일의 워크셀로 바꾸어야 한다고 저술한 바 있습니다.¹¹ **Telstra Enterprise Services**팀은 시스템을 점검하거나 업데이트를 수행하는 프로세스가 너무 번거롭고 복잡하기 때문에 개발자 그룹이 내부 시스템을 무시하도록 했습니다. 이렇게 실무 그룹을 조정함으로써 사이클을 212일에서 42일로 단축할 수 있었습니다.¹²

이 예는 인프라팀이 내부 그룹을 더 효과적으로 지원하는 데 프로세스 변화가 얼마나 중요한 역할을 하는지 보여줍니다.

애자일 통합 기술은 보다 민첩한 인프라를 뒷받침하며 API, 컨테이너 이미지, 분산형 통합은 소프트웨어 인프라 언급에서 새로운 주제로 부상했습니다.

10 Froelich, Andrew, '애자일 IT 구현에 관한 찬반 양론', 2015년 10월 6일.
<http://www.informationweek.com/infrastructure/pc-and-servers/should-it-go-agile-the-pros-and-cons/d/d-id/1322448>

11 Pearce, Ronan, '애자일 인프라는 실현 가능한가?' 2013년 6월 20일.
https://www.cio.com.au/article/465436/can_infrastructure_agile/

12 <http://agilemanifesto.org/>

애자일 선언문(Agile Manifesto)에는 소프트웨어 개발의 네 가지 핵심 원칙이 정의되어 있습니다.¹² 해당 원칙들은 애자일한 통합 기반 인프라에서 통합 전략에 적용할 수 있습니다.

1	프로세스와 툴보다 개인과 개인 간의 상호작용이 더 중요하다 인프라가 갖춰진 상태에서는 팀 간의 상호 작용에 초점이 맞춰집니다. 상호 작용에는 API, 메시징, 트래픽 패턴 등에 의해 제어되는 직접적인 커뮤니케이션, 시스템 차원의 상호 의존성, 테스트 및 릴리스 프로세스(예: CI/CD 파이프라인)가 포함됩니다.
2	포괄적인 문서보다 작동하는 소프트웨어가 더 중요하다 인프라는 그 특성상 상시 작동되어야 하며 급격하게 바뀌는 것이 아니라 서서히 조정되어야 합니다. 따라서 기본적으로 중단 없이 작동하는 인프라가 요구됩니다. 인프라 전략 측면에서 '작동한다'는 말은 인프라 구성 요소를 통해 적정 성능 범위 내에서 최종 사용자 동작이 적절하게 이루어진다는 의미입니다.
3	계약 협상보다 고객과의 협업이 더 중요하다 인프라 시스템이 갖춰지면 인프라팀이 시스템 의존성을 관리하는 방법(예: 보안 정책, 서비스 수준 계약, 게시된 API)이 계약서에 명시되며 이 시스템의 내부 및 외부 사용자가 모두 고객이 됩니다. 민첩성이 구현되면 시스템에 관련된 정책과 인터페이스가 변경될 때 사용자가 이를 알 수 있을 뿐 아니라, 이러한 변경을 더 신속하게 적용할 수 있습니다. 분산형 통합을 사용하면 여러 팀이 통합을 개발하고 배포할 수 있게 되어 협업의 범위가 확장됩니다.
4	계획을 따르기 보다 변화에 대응하는 것이 더 중요하다 이 원칙에 기반을 두고 기술을 통해 프로세스를 지원할 수 있습니다. 인프라를 위해 시스템의 안정성은 계속 유지하되, 컨테이너와 같은 신기술을 사용하여 탄력적인 플랫폼을 제공해야 합니다. 요구에 따라 인스턴스를 동적으로 추가하거나 삭제하고, 배포 및 업데이트를 자동화하며, 여러 인스턴스에서 변경을 오케스트레이션할 수 있습니다. 게시된 API 정의를 통해 재사용 가능한 툴을 사용하면 개발에서 일관성이 유지되며 이 접근 방식을 기반으로 변화에 적용할 수 있는 안정적인 플랫폼을 구축할 수 있습니다.

그림 6. 애자일 선언문(Agile Manifesto)에 명시된 소프트웨어 개발의 핵심 원칙

애자일 통합 아키텍처는 기술을 통해 인프라팀 내의 문화가 바뀌도록 지원하여 인프라 전략을 위한 기반을 구축합니다. 이로 인해 인프라 기술과 팀이 개발 및 비즈니스 전략에 더 긴밀히 연계되도록 할 수 있습니다.

애자일 방법론을 사용하면 개인, 빌드, 종속성과 같은 소프트웨어 프로젝트의 핵심 요소를 식별한 후 이들 요소 간의 관계를 정의할 수 있습니다. 애자일 프로젝트로서 통합 인프라에 접근하는 경우 팀, 컨테이너 이미지, API, 통합 지점 등 애자일로 정의되는 유사 요소와 관계가 존재한다는 것을 발견할 수 있습니다. 표 3에서는 이러한 유사 요소 중 일부를 설명합니다.

표 3. 소프트웨어 애자일과 인프라 애자일 요소 비교

프로젝트	조직	설명
개인	팀	팀은 인프라의 특정 부분을 담당합니다. 팀에서 관리하는 시스템과 API, 팀 리더, 팀 목표 등 팀이 담당하는 업무와 관련된 정보를 식별합니다.
모듈	API	인터페이스(API)가 명확히 정의되어 있고 장기간 안정적이며 자체 로드맵을 보유하여 특정 팀에 의해 실행됩니다. 또한 조직 내에서 중요한 특정 기능을 구축합니다.
빌드	컨테이너 이미지	릴리스는 테스트를 거쳐 태그가 지정된 배포 가능 유닛을 기반으로 하며 액세스 권한을 가진 팀이 안정적으로 배포할 수 있습니다. 이는 모놀리식 버전 코드를 대체합니다.
컴파일 종속성	통합	이 요소는 이러한 분산 시스템 내부의 다양한 구성 요소 간 통합과 매핑을 식별합니다. 그런 다음 시스템의 다른 부분처럼 통합 지점에 대해 관리, 커미셔닝, 디커미셔닝, 버전 관리, 테스트를 수행합니다.
빌드 테스트	인프라 자동화	소프트웨어 빌드, 성능 및 사용자 요구 사항을 테스트하는 기능부터 여러 시스템을 운영하고 모니터링하는 작업에 이르는 전체 라이프 사이클 관리입니다.

애자일 원칙을 인프라 계획에 적용

대부분의 변경 관리 접근 방식에서는 모든 하위 시스템을 종합적으로 문서화해야 합니다. 이 문서는 모니터링 방법에서 성능 매개 변수 그리고 담당 팀에 이르기까지 시스템의 모든 측면을 상세하게 다루어야 합니다. 애자일 원칙에서는 협력과 적응성이 요구되며 이는 많은 문서화 작업을 필요로 하는 변경 관리와 충돌합니다.

잠재적인 이해 관계자, 변경 사항, 시스템 구성 요소 전체를 규범적으로 정의하려 할 것이 아니라 변경 요청과 계획을 평가하는 데 사용할 수 있는 일련의 지침과 표준을 정의해야 합니다. 다음 질문에 대한 답을 생각해 보십시오.

- 사용자를 위해 설계된 엔드 투 엔드 환경은 무엇인가?
- 각 팀, API, 시스템 등이 시간이 지남에 따라 이러한 환경을 개선하기 위해 어떻게 기여하고 있는가?
- 모니터링과 경고는 어떤 매개변수로 어떻게 정의되며 서비스 수준을 어떻게 유지하는가?
- 작동이 제대로 되는지 확인하기 위해 어떤 유형의 자동화된 테스트가 필요한가?
- 사용자 환경을 방해하지 않으면서 팀이 자체 하위 시스템의 새 버전을 테스트하고 배포하기 위한 릴리스 파이프라인은 무엇인가?
- 구성 요소 서비스의 장애가 전체 시스템의 서비스 수준에 어떤 영향을 미치는가?

애자일 인프라 내에서의 변경 관리는 계약보다는 지속적인 협력을 통해 이루어져야 합니다.

목표를 달성할 확률

귀사의 IT 프로젝트가 성공할 가능성은 어느 정도일까요? 먼저 사양을 충족하는 것, 고객 채택률을 높이는 것, 출시를 완료하는 것 등 성공의 기준을 파악하는 것이 중요합니다. 프로젝트 관리 교육을 전문으로 하는 그룹인 4PM은 예산 범위 내에서 사양에 맞게 정시에 프로젝트를 완료하는 것을 성공이라 정의합니다.¹³ 이러한 정의를 기반으로 이 그룹은 IT 프로젝트의 약 70% 정도는 실패한다고 추정합니다.¹³ 이러한 수치가 변화가 일어나고 있습니다. 최근에 실시된 Project Management Institute 설문조사에 의하면 지난 5년간 계획한 목표를 달성한 프로젝트의 수가 증가했음이 밝혀졌습니다.¹⁴ 이렇게 수치가 증가한 것은 IT팀과 실무팀이 밀접하게 연계되어 자사의 전략에 대해 그리고 고객에 대해 더 나은 정보를 활용할 수 있기 때문이라고 합니다.⁸

이렇게 전략적 연계를 구축하는 이유 중 하나는 애자일 팀을 구현하는 것입니다. 애자일 방식은 협력과 피드백, 시스템과 문제에 대한 전체적인 시각 및 창의적인 접근 방식을 권장합니다.

¹³ 4PM.com, '프로젝트가 자주 실패하는 이유' 2015년 9월 27일.

<http://4pm.com/2015/09/27/project-failure/>

¹⁴ Florentine, Sharon, 'IT 프로젝트의 성공률, 마침내 증가하다.' 2017년 2월 27일.

<https://www.cio.com/article/3174516/project-management/it-project-success-rates-finally-improving.html>

공유된 기술 스택이 있으면 개별 코드에서 벗어나 시스템과 상호의존성을 논의할 수 있게 됩니다. 즉, 시스템 수준의 사고를 하면서 내부에서 개발한 소프트웨어, 벤더 시스템 그리고 이들 간의 연결을 포함한 전체 소프트웨어 인프라 컬렉션을 단일 시스템으로 다루는 것입니다. API와 메시징 시스템은 전체 인프라를 포괄하여 소프트웨어 시스템을 통합할 수 있습니다.

API와 분산형 통합은 개발팀 또는 운영팀 내에서 개발되고 파악될 수 있으므로 통합과 관련하여 팀의 책임을 더 명확히 알 수 있습니다. 개발과 배포를 처리하는 팀이 시스템과 애플리케이션 간의 상호의존성을 인식할 수 있기 때문에 통합 자체를 더 면밀히 파악할 수 있게 됩니다.

인프라를 위한 기반으로 통합을 사용하고 이 통합과 관련된 책임을 여러 팀에 분산하면 애자일 접근 방식을 적용하기 알맞은 인프라 환경이 구축됩니다.

결론: 애자일 통합 실현

민첩성은 프로젝트가 아니라 프로세스입니다.

시장 변화에 대응하는 능력이 오늘날처럼 중요한 때도 없었으며 이를 위해서는 IT 시스템이 변화를 인식하여 새로운 서비스를 출시하거나 기존 서비스를 빠르게 업데이트할 수 있어야 합니다. 디지털 서비스의 기반이 되는 IT 인프라를 재고하는 것도 아주 중요한 사항이 되었습니다.

지금까지 인프라팀은 위험 완화와 안정성 유지라는 요구 때문에 시간이 많이 소요되는 모듈레이티드(Modulated) 프로세스에 얽매어 있었습니다. 하지만 인프라를 보는 사고방식을 하드웨어 또는 플랫폼 기반에서 통합 기반으로 전환할 수 있습니다. 통합은 인프라의 일부가 아니라 데이터, 애플리케이션, 하드웨어, 플랫폼이 포함된 인프라라는 개념의 접근 방식입니다.

통합 기술을 사용하여 민첩성과 적응성이 더 높은 인프라를 구축하는 접근 방식을 애자일 통합이라고 정의합니다. 애자일 통합에는 다음과 같은 세 가지 핵심 요소가 있습니다.

- **분산형 통합:** 메시징과 엔터프라이즈 통합 패턴을 사용하여 데이터와 시스템을 통합합니다. 이는 소규모의 팀 기반 통합으로 세분화되며 필요에 따라 여러 프로젝트와 점점에 걸쳐 분산됩니다.
- **내부 API 관리:** 재사용 가능한 인터페이스 세트를 구축하여 개발팀이 애플리케이션과 시스템에 연결될 수 있도록 합니다. API는 애플리케이션의 상호 작용을 위한 지침과 구조를 제공합니다.
- **컨테이너:** 통합 프로젝트를 개발 및 운영 프로젝트와 긴밀히 연계할 수 있으며 DevOps를 사용한 소프트웨어 프로젝트처럼 통합을 개발하고 테스트하여 출시할 수 있습니다.

E-BOOK 애자일 통합: 엔터프라이즈 아키텍처를 위한 청사진

기술은 문화의 변화를 지원하는 데 사용해야 합니다. 이는 소프트웨어뿐 만 아니라 인프라팀이 더 민첩해지도록 노력하는 것을 의미합니다. 인프라팀은 애자일 원칙을 적용하면서 기술을 서서히 도입하여 이러한 변화를 지원할 수 있습니다. 하나의 프로젝트에서 전체 조직을 근본적으로 재구성하여 민첩하게 만들 수는 없습니다. 하나의 애자일 통합 기술을 구현하거나 비즈니스의 한 영역을 변경한 후 변화를 점진적으로 확대해 나가는 것이 효과적입니다.

변화에 대한 IT 인프라의 대응력을 높이는 것은 장기적이고 전략적인 목표입니다. 이러한 목표를 달성하기 위해 조직 전체를 아우르는 전면적인 변화가 필요한 것은 아니며 격리된 상태에서 변화를 적용한 후 돌아오는 것조차 필요하지 않을 수 있습니다.

애자일 통합은 IT 인프라를 재구성하는데 필요한 기술적이면서도 조직적인 프레임워크를 제공합니다.

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 어워드와 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