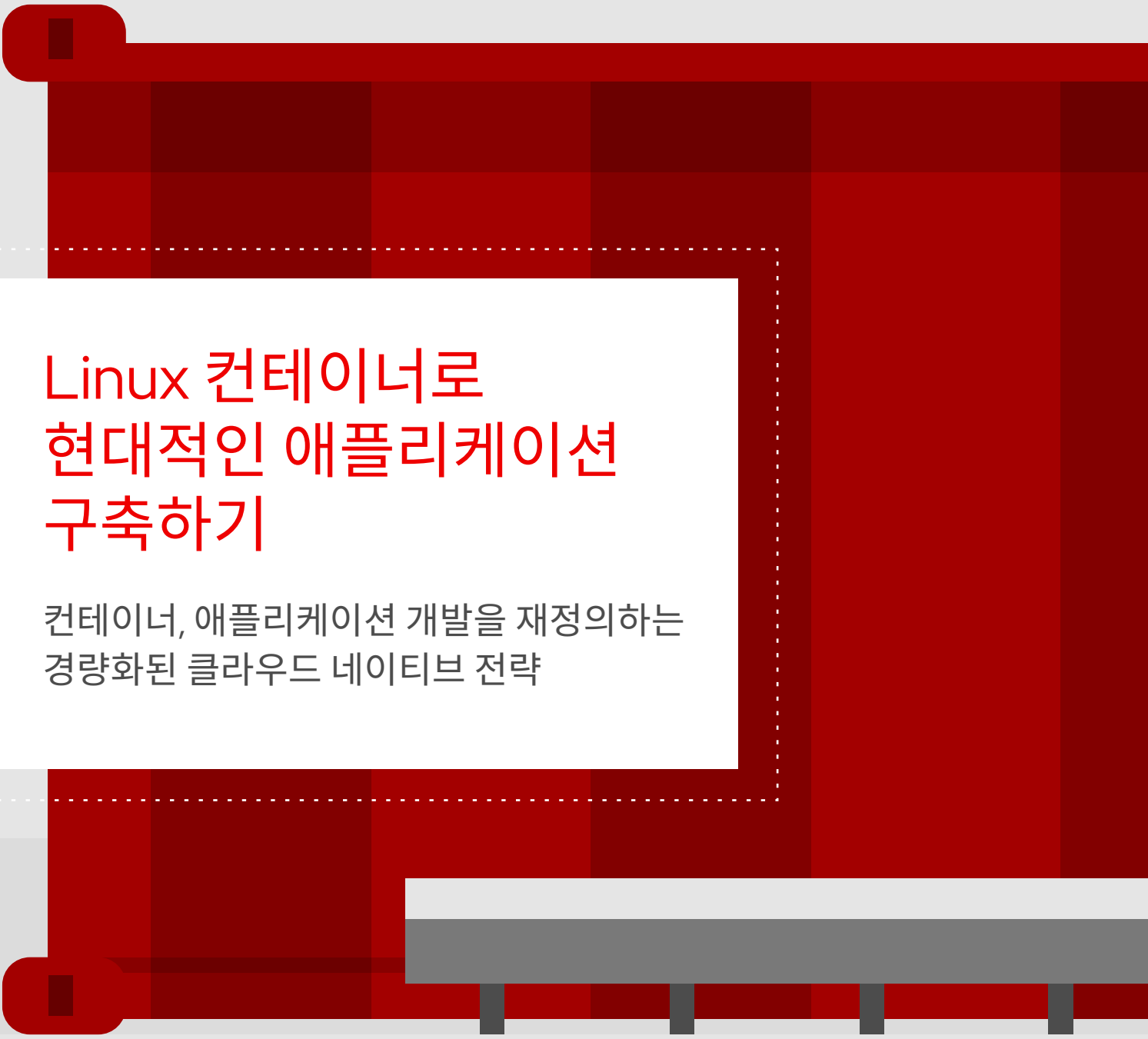
Red Hat

Linux 컨테이너로 현대적인 애플리케이션 구축하기

컨테이너, 애플리케이션 개발을 재정의하는 경량화된 클라우드 네이티브 전략



목차

소개

컨테이너 안에는 무엇이 있을까요?

모놀리식 애플리케이션 개발의 시대는 지났습니다. "디지털 트랜스포메이션"이 블록체인, 애자일, 클라우드와 함께 따라오는 최신 슬로건처럼 들릴 수 있지만, 과장된 유행어는 아닙니다. 트랜스포메이션을 통해 전혀 새로운 차원의 속도, 일관성 및 효율성을 구현하면서 개발자의 작업 방식이 근본적으로 변화하고 있기 때문입니다.

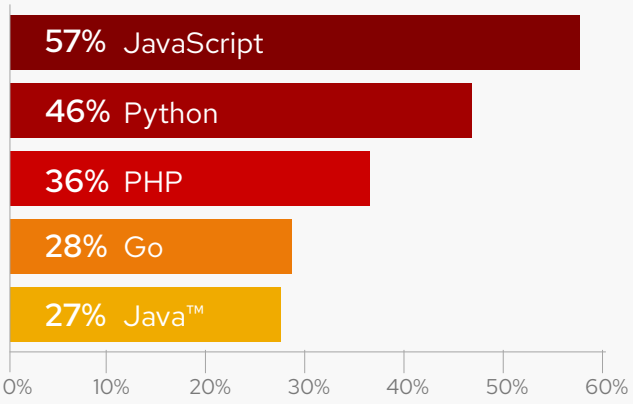
귀사도 예외는 아닙니다.

IT 개발자는 비즈니스 목표를 실현하기 위한 부담과 압박감을 갖게 됩니다. 컨테이너 기술은 사용자가 실시간으로 모든 기기에서 새로운 애플리케이션과 기능, 업데이트를 사용할 수 있도록 지원하는 비밀 병기입니다. 컨테이너는 일관된 개발 환경을 구축하여 어디서나 실행 가능한 클라우드 네이티브 애플리케이션을 신속하게 개발해 제공하므로 더욱 효율적으로 업무를 수행할 수 있습니다. 컨테이너를 활용하면, 마이크로서비스를 제공하여 긴 회귀(regression) 테스트 주기를 없애고, 중단없이 배포하며, 기능별로 코드를 패치하거나 롤백하기 위한 메커니즘을 제공할 수 있습니다.

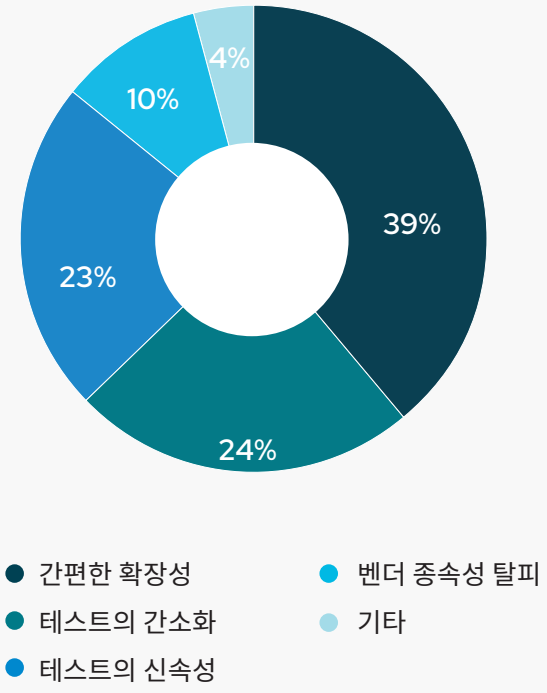
컨테이너화된 애플리케이션의 배포 방법을 알아보세요.
교육 과정 등록하기 ▸

컨테이너는 상대적으로 새로운 기술이지만, 전 세계 개발자들은 개발을 현대화하기 위한 유용하고 기본적인 툴로서 컨테이너를 이미 사용하고 있습니다. 컨테이너 기술이 빠르게 완성화되고 컨테이너 오케스트레이션 및 관리 툴이 발전함에 따라 컨테이너 도입이 정점에 도달하였고, 2018 Digital Ocean 리포트에 따르면 현재 컨테이너를 사용하는 개발자 비율은 49%에 이릅니다.[1]

컨테이너에서 사용되는 개발 언어[1]



컨테이너의 이점[1]



1 *Digital Ocean. Currents. 2018년 6월.*

컨테이너의 역사

과정이 곧 결과가 되어온 여정

"요하네스 구텐베르크의 활판 인쇄술이 발명된 후 사람들이 안경을 많이 쓰기 시작했는데, 사람들이 책을 더 많이 접하게 되면서 이전엔 몰랐던 사실, 즉 자신이 눈이 나쁘다는 것을 깨닫게 되었습니다. 이로 인해 안경의 시장 수요가 늘어나자 렌즈를 생산하고 개발하는 사람들의 수도 늘었습니다. 이 연결고리에서 현미경이 탄생되었고, 곧이어 인체가 아주 작은 세포로 이루어져있다는 사실을 발견하게 되었습니다."

Steven Johnson
우리는 어떻게 여기까지 왔을까: 현대 사회를 만든 6가지 혁신(How We Got to Now: Six Innovations That Made the Modern World)

여러 가지 다른 뛰어난 기술적 발전과 마찬가지로, 컨테이너는 시간이 지나면서 진화를 거듭해 온 몇 가지 개념과 기술의 정점이라 할 수 있습니다. 1970년대와 80년대부터 코드를 오브젝트로 나누고 추상화(abstraction)와 격리(isolation)의 개념을 시험하기 시작했습니다. 코드의 일부를 보호하고 다른 일부를 노출했을 때, 프로세싱과 데이터 처리에 대한 제어 능력이 향상되어 인접한 시스템을 유연하게 통합할 수 있다는 것을 배웠습니다. 이러한 발전을 통해 프로세스 및 구성 요소를 레이어화하고 추상화하는 데까지 나아갔으며, 다중 계층 환경과 서비스 지향 아키텍처(Service-Oriented Architecture, SOA)로 진화하여 비즈니스 코드 및 사용자 인터페이스에서 데이터 레이어를 더 멀리 격리할 수 있었습니다. 이 시기에 모놀리식 워터폴 개발로부터 소프트웨어 개발 라이프사이클(SDLC)을 거쳐 애자일 개발과 스크럼, 마침내 DevOps 및 지속적 제공의 시대로 방법론이 진화되어 왔습니다.

비즈니스 관점에서 보면, 이 모든 발전은 코드를 더 빠르고 경제적이며 더 우수하게 생성할 수 있는 개발 프로세스를 경량화하는 데 일조했습니다. 개발자 관점에서 보면, 새로운 혁신이 반복될 때마다 개발 주기가 단축되었고 더욱 엄격한 패턴 및 방법론 준수가 요구되었습니다. 모든 발전은 올바른 방향으로 나아가는 한 걸음이었듯이, 컨테이너의 출현은 모든 요소를 통합하고 진정한 유연성, 상호운용성 및 이식성을 코드에 부여하는 솔루션을 제공했습니다.

특히 Linux® 컨테이너의 진화 과정을 보면 지난 15~20년간 그 영향력이 얼마나 컸는지 알 수 있습니다.

2000
처음 프리BSD 제일(FreeBSD Jail)로 등장한 컨테이너 기술로 인해, 전체 시스템에 대한 작업 없이 개발자가 작업하는 하위 시스템으로 서버를 분할하는 것이 가능해졌습니다.

2001
컨테이너라는 개념은 한개의 박스에서 여러 개의 범용 Linux 서버를 실행하는 목적의 Linux-VServer 프로젝트를 통해 Linux로 옮겨갔습니다.

2007
기술을 더 추가하고 결합하여 컨테이너화가 더욱 구체화되었습니다. 특히 제어 그룹(cgroups), systemd 및 커널/사용자 네임스페이스는 전반적인 제어 및 가상화 기능을 추가했으며 이는 환경을 분리하기 위한 프레임워크 역할을 했습니다.

2008
Docker 및 컨테이너 기술로 인해, 더 많은 개념과 툴링이 추가되어, 사용자가 신속하고 새롭게 계층화된 컨테이너를 구축하고 이를 다른 사용자와 공유할 수 있게 되었습니다.

2012
마이크로서비스 아키텍처는 유연하고 독립적이며 배포 가능한 소프트웨어를 구축하는 데 사용하는 SOA를 전문화하고 구체화하면서 발전했습니다.[4]

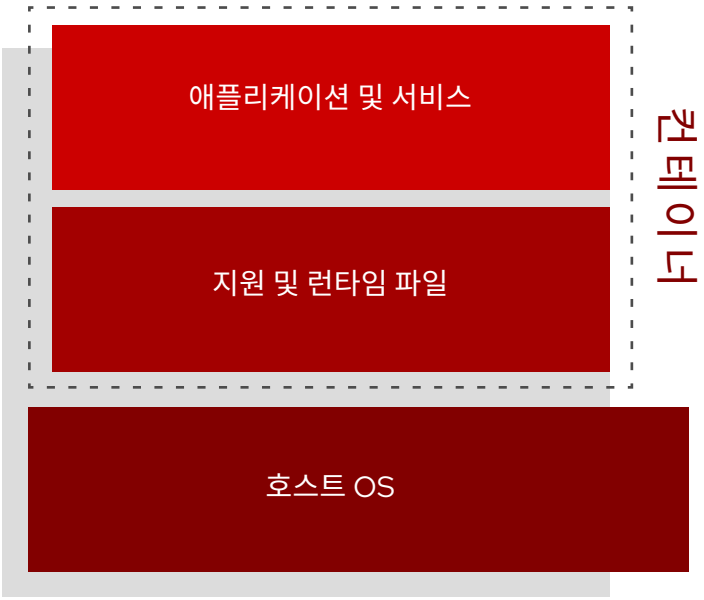
2015
쿠버네티스가 애플리케이션 배포 자동화, 확장 및 관리를 위한 오픈소스 컨테이너 오케스트레이션 시스템으로 출시되었습니다.

4 *Fowler, Martin. "마이크로서비스: 새로운 아키텍처 용어의 정의(Microservices: a definition of this new architectural term)." 2014년 3월.*

1장: 베이직

Linux 컨테이너란?

핵심을 짚으면, 컨테이너는 간단히 말해서 하나 이상의 프로세스를 시스템의 나머지 부분에서 추상화하는 새로운 방법입니다. 컨테이너는 로드(load)를 줄여주므로, 전반적인 런타임 환경에 영향을 주지 않고 코드의 소규모 서브세트 작업을 수행할 수 있습니다. 또한 애플리케이션 코드, 설정 및 종속성을 단일 오브젝트로 패키징하여 격리하는 표준 방식을 제공합니다.



성능, 속도, 가격 경쟁력 모두 향상

컨테이너의 진정한 가치는 이식성에 있습니다. 컨테이너화된 애플리케이션, 기능 또는 구성 요소를 실행하는 데 필요한 모든 파일이 단일한 고유 이미지에 존재하므로 Linux 컨테이너는 개발에서 테스트, 프로덕션에 이르는 전 과정에서 일관성과 예측 가능성을 제공합니다. 그렇기 때문에 컨테이너 배포는 복제된 개발, 테스트 및 프로덕션 환경에 의존하는 모놀리식 개발 파이프라인에 비해 훨씬 빠르고 안정적이며 경제적입니다. 뿐만 아니라 추가 개발 시간과 오랜 시간이 걸리는 테스트 주기, 또는 특정 배포 프로세스 없이 컨테이너화된 코드를 한번 작성해 개발하여 이를 다중 운영 환경에 배포할 수 있습니다.

컨테이너는 서버에 설치된 운영 체제(Operating System, OS)를 공유하고 리소스가 격리된 프로세스 형태로 실행되므로, 환경에 상관없이 빠르고 신뢰할 수 있으며 일관된 배포가 보장됩니다.

> 컨테이너의 진정한 가치는 **이식성**에 있습니다.

가상화 vs 컨테이너

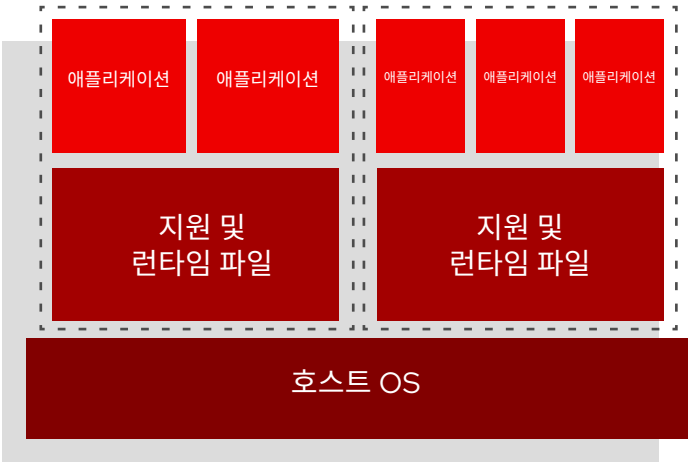
컨테이너와 가상화는 비슷해 보이지만 실제로는 매우 다릅니다.

가상화

가상화는 서로 다른 여러 대의 컴퓨터를 단일 하드웨어에서 실행하도록 지원합니다. OS 및 해당 애플리케이션은 단일 호스트 서버의 하드웨어 리소스를 공유합니다. 각 가상 머신(VM)은 자체 OS 기반을 필요로 합니다. 하이퍼바이저는 이 VM을 생성하여 실행합니다.

VS.

컨테이너



컨테이너는 애플리케이션 프로세스를 시스템의 나머지 부분에서 격리하고 일부 OS 파일, 지원 프로그램 및 라이브러리, 시스템 리소스 등 특정 애플리케이션 실행에 필요한 요소만 포함합니다. 컨테이너는 경량이며, VM보다 훨씬 빠르게 시작할 수 있고, VM 메모리의 일부만 사용합니다.

컨테이너 + 마이크로서비스 = 강력한 조합

지금까지 컨테이너의 정의를 설명했고, 이제 **개발자에게 가장 중요한 컨테이너 사용 방식인 마이크로서비스 개발**에 대해 알아보겠습니다.

마이크로서비스는 소규모의 자립적인 단일 기능 애플리케이션으로, 애플리케이션 프로그래밍 인터페이스(API)를 통해 통신합니다. 마이크로서비스 아키텍처의 핵심 원칙은 각 마이크로서비스가 한번에 단 하나의 기능만을 처리하며, 코드의 내부와 외부로 통신을 허용하는 잘 정의된 API를 제공하는 점입니다. 마이크로서비스는 궁극적인 캡슐화 메커니즘입니다. 마이크로서비스는 완전하게 자립적이기 때문에, 마이크로서비스를 변경하더라도 모놀리식 코드 구조에서 코드를 변경할 때에 비해 전반적인 애플리케이션에 리스크 발생 위험이 줄어듭니다. 또한 마이크로서비스는 자립적이며 시스템 리소스를 별도로 사용하기 때문에 전통적인 애플리케이션보다 더 빠르고 민첩합니다.

컨테이너와 마이크로서비스는 독립적으로 존재할 수 있으며 실제로도 그런 경우가 많습니다. 개별적으로 보면 이 둘은 용도가 다르지만, 함께 구현되는 경우에는 이식성을 갖춘 클라우드 네이티브 애플리케이션을 구축하는 강력한 툴이 됩니다.

마이크로서비스 지원 기술로서의 컨테이너를 생각해 보세요. 컨테이너는 호스트 OS로부터 추상화되며 컨테이너에 포함된 코드를 실행하는 데 필요한 모든 지원 및 런타임 파일을 포함합니다. 컨테이너를 배포하면 기반이 되는 OS에 상관없이 실행됩니다. 컨테이너는 이식성이 있으므로 재구축이나 추가 테스트 없이 여러 클라우드와 기기에 배포할 수 있습니다.

컨테이너화된 배포 방식으로 마이크로서비스를 개발하는 것이 엔터프라이즈 개발의 표준으로 자리잡고 있습니다. 아키텍처가 개발자의 다양한 일상 업무에 전례 없이 새로운 차원의 민첩성, 속도, 리소스 효율성을 제공합니다. 개발자는 DevOps 환경에서 컨테이너와 마이크로서비스를 사용해 각 서비스를 독립적으로 배포할 수 있습니다. 이러한 프랙티스를 활용하면 코드 변경 사항을 병합할 필요가 없어지고 테스트를 대폭 개선하여 테스트 및 프로덕션 환경 모두에서 결함 격리를 지원할 수 있습니다. 또한 이와 유사한 개발자 팀은 탄력적으로 결합된 애플리케이션 작업을 수행하고 팀 내 요구 사항을 다른 팀에 강요하지 않고도 그에 가장 적합한 기술 스택을 선택할 수 있습니다.

컨테이너 이미지로 재사용, 재활용, 비용 절감

개발자들이 진정으로 사랑하는 단 한가지가 있다면, 그것은 바로 코드를 재사용하는 것입니다. 컨테이너를 활용하면 베이스 컨테이너 이미지를 생성하고 이를 리포지토리에 추가하여 새로운 프로젝트를 시작할 때마다 가져올 수 있습니다. 베이스 컨테이너 이미지는 실행 파일을 포함하지 않는, 변경 불가능한 정적 파일이기 때문에 일관성과 이식성을 유지하여 어떤 인프라에서든 격리된 프로세스를 실행할 수 있습니다. 이미지는 애플리케이션을 실행하기 위해 필요한 시스템 라이브러리, 시스템 툴 및 기타 플랫폼 설정으로 구성됩니다.

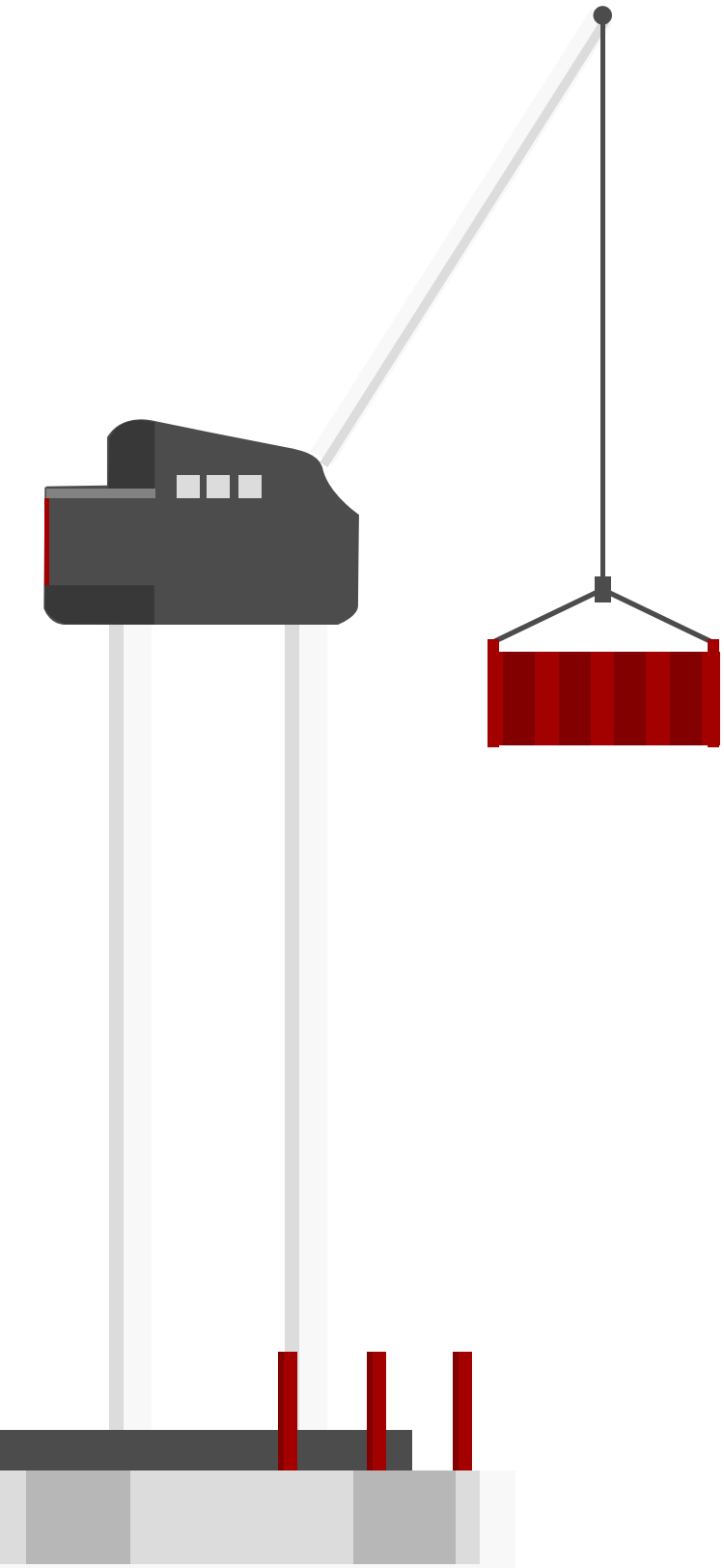
개발자는 자체 컨테이너 이미지를 생성하거나 사용 가능한 퍼블릭 리포지토리에서 선택할 수 있습니다. Red Hat과 Microsoft를 비롯한 다양한 소프트웨어 벤더들은 자사 제품의 공용 이미지를 생성합니다.

마이크로서비스와 컨테이너는 특히 DevOps 환경에 통합될 때 강력한 시너지 효과를 냅니다.

컨테이너 이미지의 5가지 주요 이점

1. 빌드 및 배포 프로세스 자동화
2. 위치 및 다운로드 편의를 위해 태그 지정
3. 취약점 검사 간소화
4. 코딩 표준 및 정책 시행
5. 시간 절약 및 재사용 권장

2장: 생산성 향상



컨테이너와 개발자

컨테이너는 애플리케이션 구축, 테스트, 배포 및 재배포를 위한 간소화된 접근 방식을 제공하므로, 간단한 프로젝트에서 미션 크리티컬 애플리케이션에 이르는 모든 경우에 적합합니다. 모든 IT 팀 및 비즈니스를 위한 컨테이너에 몇 가지 이점이 있지만, 개발자가 이를 통해 일상 업무의 효율을 높이는 방법을 살펴보겠습니다.

유연성 확보

개발 주기를 생각해보면 시간이 충분한 경우는 거의 없습니다. 비즈니스는 로드맵을 가지고 있으며, 테스트 팀에는 반복되는 결함이 누적되어 있고, 운영 팀은 시스템에 지속적으로 패치를 적용해 보안을 유지하기 위해 노력하며, 고객은 중요 사항에 대한 자체 의견을 갖고 있습니다. 안타깝게도 주어진 시간은 한정되어 있으며 특정 프로젝트를 수행하는 개발자의 수도 한정되어 있다는 사실입니다. 애플리케이션이 컨테이너 기반에서 실행되는 경우 개발자와 비즈니스 및 운영 팀은 변경 로그, 결함, 보안 문제, 패치 수준 및 새로운 고객 기능 요청에 실시간으로 대응할 수 있습니다. 뿐만 아니라 애플리케이션 아키텍처의 나머지 부분에 영향을 주지 않고 단일 컨테이너를 배포할 수 있기 때문에 옆 팀이 개발 주기를 마칠 때까지 기다릴 필요 없이 즉시 변경할 수 있습니다.

폭넓은 사고

컨테이너는 경량이며 몇 밀리초 내에 시작할 수 있는 경우가 많습니다. OS 부팅이 필요 없으며 필수적인 디펜던시만 로드하고, 단 몇 초만에 컨테이너를 생성, 복제 또는 제거할 수도 있습니다. 고객이 변화하는 시즌에 따른 로드를 생성하거나 비즈니스가 새로운 사용자 팀을 추가하기로 결정하는 경우, 운영 팀은 필요에 따라 컨테이너가 사용할 리소스를 추가하여 적합하게 대응할 수 있습니다. 또한 컨테이너화된 애플리케이션을 새로운 클라우드로 배포하기만 하면 글로벌 규모의 새로운 사용자로 확장할 수도 있습니다.

효율적인 업무

컨테이너를 사용하면 특정 OS 버전과 애플리케이션 설정을 고민할 필요 없이 애플리케이션 로직에 집중할 수 있습니다. IT 운영 팀의 본래 업무에 주력하게 되는 셈입니다. 게다가 컨테이너로 코드, 종속성 및 설정을 캡슐화된 단일 요소로 패키징할 수 있으므로 손쉽게 버전 관리, 테스트 및 배포가 가능합니다. 컨테이너를 서비스 기반 아키텍처와 결합하면 애플리케이션을 보다 간편하게 지원하고 테스트하여 성능을 개선할 수 있습니다.

(실질적인) 표준화 달성

개발에서 프로덕션에 이르는 모든 환경을 표준화하는 것은 이상적으로 들일 수 있습니다. 하지만, 컨테이너의 가장 강력한 장점들 중 하나는 바로 컨테이너가 로컬, 개발, 테스트, QA 및 프로덕션 환경을 표준화하는 점입니다. 높은 수준의 예측 가능성을 통해, 격리된 환경을 스핀업하고, 디버깅 시간을 단축하며, 패치 수준과 운영 체제 및 애플리케이션의 차이로 발생하는 문제를 진단할 수 있습니다. 이로 인해, 새로운 기능의 개발과 제공에 더 많은 시간을 투자할 수 있습니다. 또한, 팀 내 새로운 개발자는 로컬 개발 환경을 설치하고 설정할 필요가 없어 더욱 빠르게 작업을 시작할 수 있습니다. 리포지토리에서 컨테이너 이미지를 가져와 코딩을 시작하면 되기 때문입니다.

WORA(Write Once, Run Anywhere)

일단 작성하면 어디서나 실행할 수 있다는 뜻의 "WORA(Write Once, Run Anywhere)"는 애플리케이션을 개발하는 OS와 크게 관련되어 있습니다. 타겟 시스템이 동일한 OS를 실행하는 경우에 애플리케이션을 어디서나 실행할 수 있다는 뜻이기 때문입니다. 컨테이너를 사용하면 개발 환경에 상관없이 사실상 모든 환경에서 코드가 실행될 수 있습니다. 컨테이너 내부에서 개발하는 경우는 베어 메탈, 가상 머신, 퍼블릭 클라우드, 프라이빗 클라우드 및 하이브리드 환경 전반에서 Linux, Windows, Mac OS로 배포할 수 있습니다. 또한 Docker 오픈소스 프로젝트가 널리 도입되면서 컨테이너 내에서 안정적인 방법으로 애플리케이션 배포를 자동화할 수 있게 되어, 다음 개발 주기를 시작하는 데 걸리는 시간을 절약할 수 있습니다.

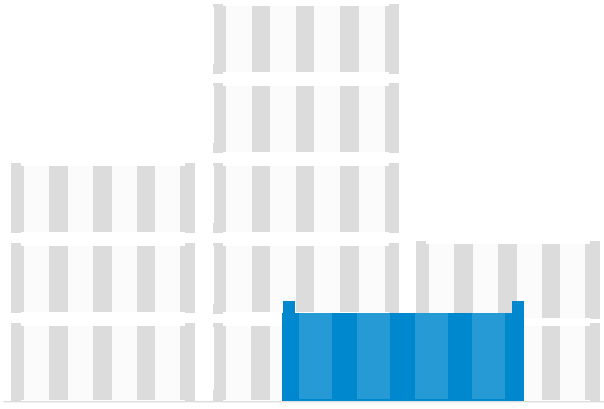
뛰어난 애플리케이션 품질 제공

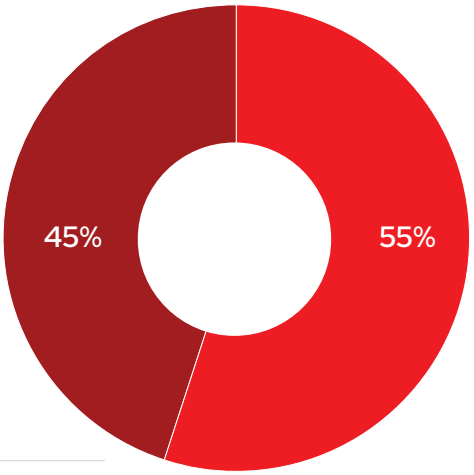
컨테이너는 누구든지 훨씬 쉽고 빠르게 테스트하고 문제를 해결할 수 있도록 해줍니다. 개발자의 경우 단일 애플리케이션 기능에 개발을 포함할 수 있어 인접한 코드에 예상치 못한 오류가 생길 가능성이 줄어듭니다. 개발자가 컨테이너의 버전을 관리하게 되면 QA 팀은 지속적인 통합(CI) 서버에서 전체 애플리케이션을 가져와 빌드하는 대신 컨테이너 이미지에서 직접 테스트할 수 있으므로 테스트를 간소화하고 시간을 절약하게 됩니다.

컨테이너는 또한 QA 및 지원 팀이 애플리케이션 문제의 근본 원인을 보다 빠르게 식별할 수 있도록 지원합니다. API 중심의 탄력적인 코드 결합으로 인해, 예기치 못한 상황이 발생하는 경우 살펴봐야 할 상호의존성이 거의 없습니다. 엔지니어들은 신속하게 문제의 근본을 밝히고 오류가 발생한 컨테이너를 정확히 찾아낼 수 있습니다. 문제가 식별되면 컨테이너는 변경 사항을 쉽게 되돌릴 수 있게 해줍니다. 단일 컨테이너는 애플리케이션의 나머지 부분에 영향을 주지 않고 롤백할 수 있습니다. 개발자가 컨테이너를 다시 빌드해야 하는 경우, 문제 해결 시간은 줄이고 비즈니스 리더 및 고객이 원하는 기능을 개발하는 데 더 많은 시간을 투자할 수 있습니다. 컨테이너를 사용하면 당일 문제 해결이 가능하므로 고객과 관리자 모두에게 이상적입니다.

최근 IDC 연구는 다음의 사항에 주목합니다.

55%의 IT 리더는 컨테이너를 온사이트로 배포한 반면, 45%는 퍼블릭 클라우드에 컨테이너를 배포한 것으로 나타났습니다.[5]





5 *IDC 연구. "컨테이너 인프라 시장 평가: 레거시 및 클라우드 네이티브 아키텍처 연결 - 사용자 설문조사 요약(Container Infrastructure Market Assessment: Bridging Legacy and Cloud-Native Architectures — User Survey Summary)." 2018년 3월.*

선호하는 툴과 언어 사용

개발자는 프로젝트에 사용하고 싶은 툴이나 언어 선택권이 있을 때 작업을 더욱 만족스럽게 수행합니다. 컨테이너는 이렇게 개발자에게 만족할만한 유연성을 제공합니다. 어떤 애플리케이션 런타임은 특정 유형의 워크로드나 아키텍처에 더 최적화되어 있습니다. 예를 들어, vert.x는 반응형 분산 아키텍처를 권장하는데, 이는 사물 인터넷(IoT) 기기에 필요한 것과 같은 실시간 반응형 애플리케이션에 적합합니다. 해당 유형의 애플리케이션을 다른 언어로 빌드할 수 있긴 하지만 vert.x가 기본으로 제공하는 항목을 개발자가 재구성해야 합니다. 컨테이너는 언어나 툴의 제약을 받지 않기 때문에 개발자가 유연성과 제어 기능을 확보할 수 있어, 전체 애플리케이션 팀이 프로젝트에 동일한 툴이나 언어를 도입하지 않고도 하나의 애플리케이션에 적합한 툴을 선택할 수 있습니다.

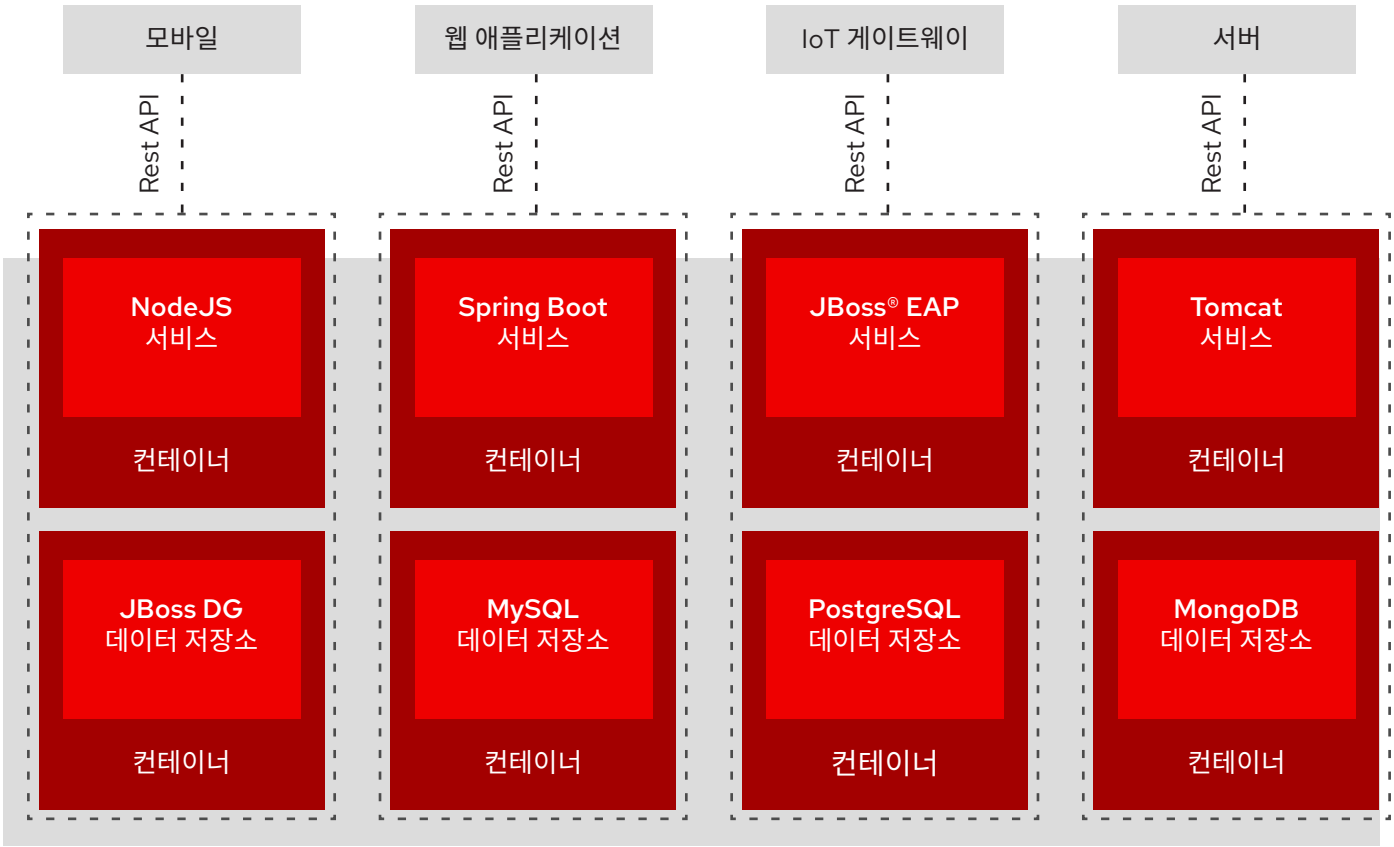


그림 1: 컨테이너로 개발 유연성 개선 및 배포 간소화

개인 역량 향상

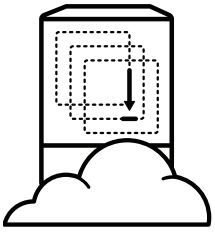
마지막 이점은 개인의 가치 역량을 높인다는 점입니다. 새로운 기술을 통해 역량을 향상하지 않으면 결국 자신의 분야에서 뒤처지거나 은퇴할 때까지 레거시 앱만 작업하게 될 수 있습니다. 성장과 개발의 기회는 성공적인 경력을 쌓는 데 있어 매우 중요하며, 대개 개발자들은 새로운 기술을 배우길 좋아하므로 직업상 혜택을 누리는 셈입니다.

마이크로서비스, 컨테이너 및 컨테이너 오케스트레이션 기술은 기술적으로 구현하기가 어려우면서도 이에 대한 수요는 매우 높은 편이라 개발자의 의욕을 불러일으킵니다. 2018 Red Hat® 설문조사 결과를 보면, 향후 2년간 컨테이너 사용이 89% 늘어나고 앞으로 대규모 도입 가속화 경향이 지속될 것으로 전망됩니다.[6] 컨테이너 기반 마이크로서비스 개발 기술을 갖추었다면 새로운 기회를 통해 역량을 개발하고 회사에 가치를 제공해보세요.

6 *Dawson, Margaret. "Red Hat 글로벌 고객 기술 전망 2019: 자동화, 클라우드 및 보안이 재정 지원 우선순위 주도(Red Hat Global Customer Tech Outlook 2019: Automation, cloud & security lead funding priorities)." Red Hat 블로그. 2018년 12월 18일.*

3장: 컨테이너 사용의 실제

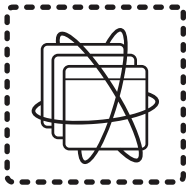
아직 컨테이너가 도입되지 않았더라도 가까운 시일 내에 코드를 컨테이너로 제공하게 될 가능성이 높습니다. 기업들이 디지털 트랜스포메이션 실험에 뛰어들면서, 경영진들도 컨테이너를 통해 IT 조직이 직원 및 장비 리소스 사용을 개선하여 비용을 절감하고 효율성을 향상하고 가치를 제공하는 방법을 배우게 되었습니다. 개발자는 프로젝트를 계획하면서 활용 사례를 염두에 두고 컨테이너에 대한 아이디어를 다음 팀 회의에서 제안해야 합니다.



리프트 앤 시프트(Lift and shift)

클라우드 네이티브 애플리케이션이 화두로 떠오르고 있지만 엔터프라이즈 애플리케이션은 여전히 모놀리식인 경우가 대부분입니다. 리프트 앤 시프트는 기존 애플리케이션을 보다 현대적인 클라우드 네이티브 아키텍처로 마이그레이션하는 프로세스를 가리킵니다. 가능한 코드 변경을 최소화하여 기존 애플리케이션을 클라우드로 간단히 마이그레이션하는 것입니다. 대체로 이러한 프로세스는 간단하며 비즈니스 리더들이 선호합니다. 기존의 미션 크리티컬 애플리케이션을 사용하여 대규모 재설계에 적극 투자하지 않고도 더 나은 가치와 성과를 창출할 수 있기 때문입니다.

리프트 앤 시프트는 간단하고 빠르긴 하지만 대체로 임시적인 솔루션으로 간주됩니다. 클라우드별 툴을 충분히 활용하지 못하기 때문입니다. 애플리케이션을 리프트 앤 시프트 방식으로 마이그레이션하는 경우, 어느 시점에서 재평가를 통해 여전히 가치 창출이 이루어지는지 확인하고 리팩토링이나 교체를 권고해야 할 수 있습니다.



리팩토링

첫 번째 사용한 애플리케이션이 구식이 된 이후에, 리팩토링은 애플리케이션 현대화에 있어 필수 요소가 되었습니다. 이는 구체적으로 컨테이너용 애플리케이션을 리팩토링하는 경우에 해당합니다. 리팩토링이 리프트 앤 시프트 마이그레이션보다 훨씬 집약적이지만, 이를 통해 애플리케이션은 컨테이너화 및 클라우드 네이티브 환경의 이점을 최대한 활용할 수 있습니다.

과거에 리팩토링 작업은 소요되는 시간과 복잡성으로 인해 간과되는 경우가 많았으나, 컨테이너는 이를 점진적인 프로세스로 만들어줍니다. 먼저 애플리케이션의 가장 중요한 부분만 수정하고 나머지는 리프트 앤 시프트 방식으로 마이그레이션합니다. 그런 다음 나머지 부분을 점진적으로 리팩토링할 수 있어, 비즈니스를 중단없이 운영하면서도 일정에 따라 추가 개선 사항을 적용할 수 있습니다. 이는 종종 마이크로서비스 관련 문서와 블로그에서 “모놀리스를 점차 축소”하는 프로세스라고 불리기도 합니다.

물론 예전 방식을 리팩토링할 수도 있지만, 그 경우에는 릴리스하여 사용하기 전에 전체 애플리케이션을 수정합니다. 두 가지 전략에는 모두 장단점이 있습니다. 점진적인 리팩토링의 주요 이점은 바로 시간입니다. 단점은 클라우드 네이티브 플랫폼으로 완전하게 마이그레이션하기는 어렵다는 점인데, 이는 관리 복잡성과 리스크를 유발합니다.

신규 애플리케이션 개발

앞서 살펴봤듯이 컨테이너는 새로운 클라우드 네이티브 애플리케이션을 개발하기 위한 강력하면서 유연한 툴입니다. 처음부터 새로 개발한다면 컨테이너의 이점을 모두 획득할 수 있습니다. 컨테이너는 마이크로서비스, 하이브리드 애플리케이션, 반복적인 작업 및 태스크 자동화, 인공지능(AI)과 머신 러닝(ML)과 같은 미래 지향적인 애플리케이션에 이상적인 플랫폼을 제공합니다.

마이크로서비스

컨테이너는 마이크로서비스 아키텍처와 밀접한 관련이 있습니다. 분산형 애플리케이션과 마이크로서비스는 구성 요소로 개별 컨테이너를 사용하여 쉽게 격리, 배포 및 확장할 수 있습니다.

하이브리드 애플리케이션

컨테이너는 코드 배포 방식을 표준화하도록 지원하므로 손쉽게 온프레미스와 클라우드 환경 간 실행되는 애플리케이션 워크플로우를 구축할 수 있습니다.

반복적인 작업 및 태스크

일괄 처리 및 추출, 트랜스포메이션, 로드(ETL) 작업과 같은 반복적인 작업과 태스크는 신속하게 작업을 시작하도록 컨테이너에서 손쉽게 개발할 수 있습니다. 그 이후 운영 편의성을 위해 이를 자동화하고 확장하여 동적으로 수요에 대응할 수 있습니다.

인공 지능(AI) 및 머신 러닝(ML)

컨테이너는 AI와 ML을 통합하는 이식 가능한 클라우드 애플리케이션을 구축하고 배포할 수 있는 새로운 방식을 제공합니다. 이러한 컨테이너는 AI와 ML 모델을 신속히 확장하여 고도의 교육 알고리즘을 처리할 수 있도록 지원합니다. 또한 AI와 ML 컨테이너는 데이터 소스 가까이에 배포하여 성능을 개선하고 교육 주기를 단축할 수 있습니다.

4장: 고려 사항 및 도전 과제

작업 간소화 – 이 장을 그냥 넘기지 마십시오

컨테이너의 영향과 성능에 대해 많은 논의가 있었으나 다른 기술과 마찬가지로 컨테이너도 고려해야 할 과제들이 있습니다. 특히 수많은 엔터프라이즈 컨테이너가 프로덕션에 배포되는 경우가 늘어나고 있기 때문에 이러한 과제를 제대로 처리하지 않으면 문제가 심각해질 수 있습니다. 지속적인 기술 발전을 비롯해 조직이 컨테이너 기반 아키텍처를 제대로 도입, 관리 및 유지하는지 주시해야 합니다.

시작하기 전에 고려할 사항

이제 새로운 프로젝트를 시작하거나 레거시 애플리케이션을 리팩토링하는 여러 아이디어가 떠오르기 시작할 것입니다. 컨테이너와 관련된 경험과 지식을 얻으려면 코딩을 직접 해보는 것이 가장 좋습니다.

하지만 시작하기 전에 먼저 몇 가지 사항을 고려해야 합니다. 컨테이너화된 애플리케이션을 구축하는 경우 전통적인 환경에서는 볼 수 없는 몇 가지 과제가 발생합니다. 다음 내용은 개발 작업에 영향을 줄 수 있는 고려 사항들을 요약한 것입니다. 이 섹션에 나온 여러 아이디어는 프로젝트 아토믹(Project Atomic)을 참고한 것입니다. 이 e-book을 끝까지 읽으셨다면, 다음의 사항을 확인해 보고 싶으실 것입니다.

데이터 전략 결정

클라우드 네이티브 애플리케이션은 대부분 스테이트리스(stateless)이지만, 많은 애플리케이션이 퍼시스턴트 데이터 스토리지를 필요로 합니다. 컨테이너에는 변경할 수 없는 스토리지가 있으므로 컨테이너가 스핀다운되면 데이터를 잃게 됩니다. 이 사실을 염두에 두고 컨테이너 상태에 관계없이 데이터가 영구적으로 유지되도록 애플리케이션을 설계해야 합니다.

컨테이너가 종료된 후 애플리케이션 데이터를 보존하는 경우 스토리지 볼륨을 컨테이너에 할당할 수 있습니다. 이렇게 할당된 볼륨은 컨테이너 상태에 관계없이 영구적으로 유지됩니다. 개발자는 애플리케이션을 공유 데이터 저장소에 쓰기가 가능하도록 설계해야 합니다. 엔터프라이즈 애플리케이션의 경우 Red Hat OpenShift® Container Storage와 같은 툴은 컨테이너 환경에 맞게 구축된 소프트웨어 정의 스토리지를 제공합니다. OpenShift Container Storage에서는 컨테이너를 추가하거나 축소하는 경우에도 데이터를 영구적으로 보관할 수 있으며, 베어 메탈, 가상, 컨테이너 및 클라우드 배포 전반에서 쉽게 확장할 수 있으므로 컨테이너를 스토리지 아키텍처로 제한하지 않고도 이식성을 한층 향상할 수 있습니다.

컨테이너 통신 활성화

분산된 애플리케이션 구성 요소는 워크플로우를 수행하기 위해 서로 통신해야 합니다. 개발자는 컨테이너 기술을 통해 상호 연결 지점을 명시하고 API를 사용해 컨테이너 간 통신 메커니즘을 제공할 수 있습니다. 이 경우 컨테이너 간(container-to-container) 통신에는 적합하지만, 데이터베이스의 경우는 어떨까요?

전통적인 데이터베이스는 대개 네트워크 상에서 소켓을 사용해 통신합니다. 컨테이너 네임스페이스가 컨테이너 상태 변화에 따라 변경되므로 이런 유형의 레거시 통신 메커니즘은 작동하지 않습니다. 컨테이너화된 애플리케이션에서는 컨테이너, 데이터베이스 및 기타 네트워크 리소스 간 네트워크 통신을 지원하는 쿠버네티스나 Red Hat OpenShift Container Platform과 같은 컨테이너 오케스트레이션 플랫폼이 필요합니다.

동기화 및 표준화

컨테이너화된 애플리케이션 중에는 호스트와 컨테이너가 일관된 동작을 위해 특정 속성을 기준으로 동기화해야 하는 경우가 있습니다. 예를 들어, 다양한 지역의 위치에서 배포된 여러 개의 컨테이너로부터 데이터를 수신하는 중앙화된 로그 서버를 생각해 볼 수 있습니다. 각 컨테이너가 서버 위치 리포트 없이 호스트와 다른 시간을 리포트한다면, 로그 타임스탬프와 정보는 운영 팀에 거의 쓸모가 없습니다. 환경에 대한 속성 세트를 동기화하고 표준화하면 중앙 데이터 저장소와 통신하는 데이터의 정확성, 적합성 및 사용성을 보장할 수 있습니다.

모든 로그 캡처

모든 애플리케이션은 문제 해결을 쉽게 만들어주는 적합한 정보를 기록해야 합니다. 애플리케이션이 로그 메커니즘에 작업, 오류 및 경고를 기록하는 경우, 사용자가 이러한 로그를 확보, 검토 및 유지할 수 있는 방법에 대해 생각하게 됩니다. 컨테이너가 네임스페이스로 분리되어 있는 데다가 직접 로컬 하드 드라이브를 비롯한 베어 메탈 구성 요소에 액세스할 수 없기 때문에, 로깅 전략을 재고해야 합니다.

로그를 수집하는 가장 쉬운 방법은 해당 작업에 맞게 구축한 툴을 사용하는 것입니다. OpenShift Container Storage와 같은 컨테이너 오케스트레이션 플랫폼은 자동으로 컨테이너 로그 데이터를 수집합니다. 그러면 개발자는 이 플랫폼이 중앙 관리 콘솔에 나타나도록 해당 데이터를 퍼시스턴트 스토리지에 저장하거나 표준 출력(std out) 또는 표준 오류(std error)로 로그를 전송하는 권장 가이드라인을 따르면 됩니다.

보안 향상

컨테이너가 사용자에게 반복적으로 문제를 일으키지 않으면서 통신할 수 있으려면 애플리케이션 컨테이너는 액세스 크리덴셜을 포함한 민감한 데이터를 전송할 수 있어야 합니다. 그러나 크리덴셜 저장은 까다로울 수 있으며 애플리케이션이 잠재적인 보안 리스크에 노출될 수 있습니다. 컨테이너에서 민감한 데이터를 전송하는 가장 흔한 방식은 비공개 환경 변수를 통한 방식입니다. 쿠버네티스 및 OpenShift Container Platform과 같은 컨테이너 오케스트레이션 플랫폼은 컨테이너화된 애플리케이션 전반에서 환경 변수를 보호하고 민감한 데이터를 전송하는 기본 메커니즘을 제공합니다.

도전 과제

기술 진화에 대한 대응

컨테이너 에코시스템의 진화와 확장은 급속도로 진행되고 있습니다. 이 에코시스템은 컨테이너 구축, 배포, 설정, 자동화 및 관리를 지원하는 툴을 포함하며, 이러한 에코시스템을 지원하는 오픈소스 프로젝트가 매우 활성화되고 있습니다. 이는 기술 발전의 미래에 긍정적이지만, 이러한 솔루션을 통해 애플리케이션을 제공하는데 필요한 기술을 유지관리하는 것은 쉽지 않습니다. 개발자는 지속적인 오픈소스 프로젝트에 활발히 참여하고 교육 과정을 수강하면서 기술 역량을 향상해야 합니다. 이를 통해 컨테이너 기술 발전 속도에 뒤처지지 않고 스스로 발전해갈 수 있습니다.

DevOps 문화 수용

본 e-book을 시작하면서 언급했듯이, 컨테이너는 애플리케이션 개발 방식을 근본적으로 바꾸어 놓고 있습니다. 따라서 애플리케이션 팀은 업무 프로세스와 문화를 변화시켜 이에 대응해야 합니다. 컨테이너를 효과적으로 사용하려면, 조직은 모놀리식 코드 모델 아래서 형성된 전통적인 분리를 제거하고 건강한 DevOps 문화를 정착시켜야 합니다. 속도와 신뢰성, 일관성을 갖추고 개발에서 배포에 이르는 보안 중심 워크플로우를 구축하려면 조직은 DevOps 원칙을 따라 이를 자동화해야 합니다. 개발자는 개발에서 배포를 넘어 고객 만족에 이르는 전반적인 프로세스의 일부가 되어야 합니다. 이제 컨테이너를 책임지고, 구축한 모든 애플리케이션 로직을 프로덕션 환경에서 실행하는 것도 개발자의 역할입니다. 실시간으로 수정 사항을 릴리스하는 기능을 갖추고, 신속하게 코드를 디버깅, 수정 및 배포하여 애플리케이션의 일부를 가능한 최적의 상태로 동작하도록 해야 합니다.

보안 유지 강화

컨테이너 작업 여부에 상관없이 보안은 애플리케이션 개발자에게 항상 과제로 남아 있습니다. 컨테이너는 애플리케이션 격리의 보안상 이점을 제공하긴 하지만 조직 내 컨테이너가 확산되면 새로운 종류의 보안 리스크가 발생합니다.

컨테이너를 사용하면 보통 애플리케이션을 더 작은 마이크로서비스로 분할해야 하기 때문에 데이터 트래픽과 복잡한 액세스 제어 룰이 증가합니다. 컨테이너 수가 늘어나면 컨테이너 간 액세스 제어가 느슨해질 가능성이 있습니다. 보안 및 액세스 프로토콜을 적합하게 준수하지 않으면 프로덕션 환경에 취약점이 발생하게 됩니다. 또한 컨테이너 이미지 리포지토리를 사용하는 조직이 많지만, 해당 이미지가 조직의 보안 및 컴플라이언스 요구 사항을 준수하는지 확인해야 합니다. 그리고 컨테이너 관련 리스크를 확실히 이해하고 항상 액세스 제어 정책을 준수해야 합니다.

관리 및 모니터링

컨테이너 배포 수가 늘어나면 컨테이너를 모니터링하기가 점점 어려워질 수 있습니다. 컨테이너가 퍼블릭 및 프라이빗 클라우드에서 실행되는 하이브리드 클라우드 솔루션을 배포하는 경우, 관리 복잡성이 대폭 증가합니다. 모든 컨테이너 및 애플리케이션에서 발생하는 이벤트를 수집, 분석 및 실행하기 위해 일관된 데이터 저장소를 구현하기 어려울 수 있으며, 일단 이벤트가 캡처되면 오류 발생 위치를 정확히 파악하기 어렵습니다. 다행히 Istio, Prometheus, Jaeger와 같은 이머징 기술이 이러한 문제를 일부 완화하는 데 도움이 될 것입니다.

결론

컨테이너화는 거의 모든 현대화된 엔터프라이즈 개발 팀에서 선택하는 방법으로, 91%의 클라우드 개발자 및 개발 관리자들이 다양한 수준으로 컨테이너를 온프레미스에서 사용하고 있습니다.[7] 컨테이너는 어디서나 실행 가능한 클라우드 네이티브 애플리케이션을 신속히 개발하고 제공할 수 있는 일관된 개발 환경을 제공하여, 더 스마트하고 효율적으로 작업할 수 있도록 지원합니다.

본 e-book은 개발자가 컨테이너의 기본 사항을 파악하고 컨테이너가 개발자 및 조직에 미칠 수 있는 영향과 그 잠재력을 이해하도록 돕습니다. 여기서 얻은 지식을 활용해 현재 프로젝트에 적용할 수 있도록 다양한 자료가 준비되어 있습니다. 이제 여러분은 애플리케이션을 아키텍처에서 분리하고 조직을 발전시키며, 최신의 민첩한 미래 지향적인 개발 프랙티스인 컨테이너화로 역량을 더욱 빠르게 향상할 수 있습니다.

지금 컨테이너를 도입하고 모든 이점을 누려 보세요. 여러분의 클라우드 네이티브 애플리케이션 개발 여정을 응원합니다.

자세히 알아보기

컨테이너화된 애플리케이션 배포 방법을 빨리 알아보고 싶으신가요? Red Hat이 제공하는 무료 온라인 과정에서 컨테이너화의 개념과 실행 과정을 확인해 보세요. 이 과정을 통해, 애플리케이션과 서비스를 컨테이너화 한 후 Docker를 사용하여 테스트하고, Red Hat OpenShift Container Platform을 사용하여 쿠버네티스 클러스터에 배포하는 방법을 알아볼 수 있습니다. 또한 OpenShift Container Platform의 S2I(Source-to-Image)를 사용하여 소스 코드에서 애플리케이션을 구축하고 배포하는 방법을 학습할 수 있습니다.

자세히 알아보고 등록하기 ▸

추가 리소스 살펴보기

Udemy
Stack Overflow
Docker
AWS
Google
Red Hat

Red Hat 소개

Red Hat은 세계적인 오픈소스 소프트웨어 솔루션 공급업체로서 커뮤니티 기반의 접근 방식을 통해 신뢰도 높은 고성능 Linux, 하이브리드 클라우드, 컨테이너 및 쿠버네티스 기술을 제공합니다. 또한 고객이 신규 및 기존 IT 애플리케이션을 통합하고, 클라우드 네이티브 애플리케이션을 개발하며, 업계를 선도하는 Red Hat의 운영 체제를 기반으로 표준화하는 동시에 복잡한 환경의 자동화, 보안 및 관리를 실현할 수 있도록 지원합니다. Red Hat은 높은 수준의 지원과 교육 및 컨설팅 서비스를 제공하여 Fortune 선정 500대 기업의 신뢰받는 조언자로 인정받고 있습니다. 또한 클라우드 제공업체, 시스템 통합 업체, 애플리케이션 벤더, 고객 및 오픈소스 커뮤니티의 전략적인 파트너로서 조직이 디지털 미래에 대비할 수 있도록 지원하고 있습니다.

7 *Evans Data Corporation. "클라우드 개발 설문조사 2017, Vol.2(Cloud Development Survey 2017, Volume 2)."*