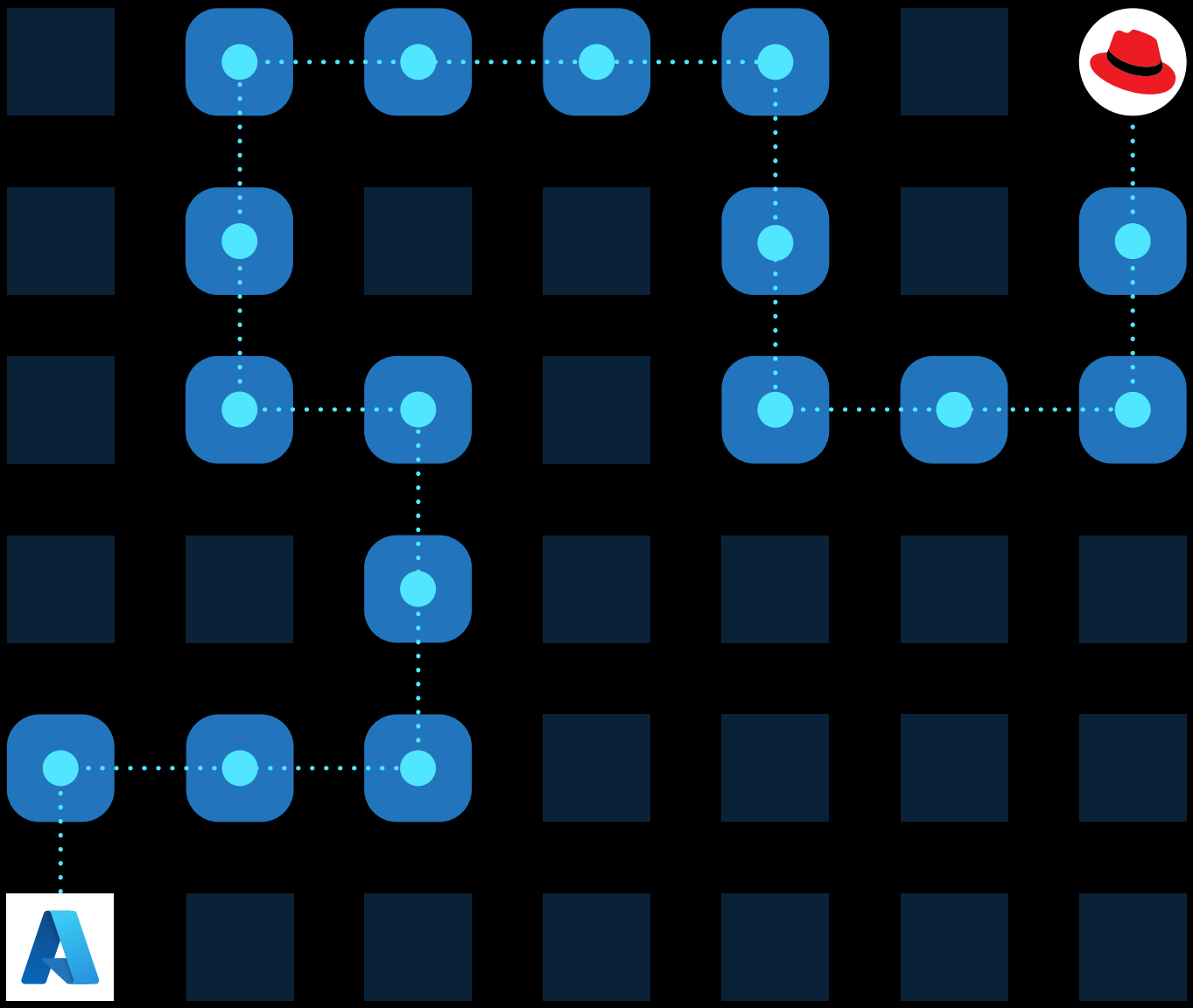
E-book 시리즈

Microsoft Azure

Azure Red Hat OpenShift 시작하기

기술 검증(POC)에서 프로덕션까지



Azure Red Hat OpenShift 시작하기

3 /

1장
Microsoft 및 Red Hat과 상담하기

6 /

2장
Red Hat OpenShift 소개

13 /

3장
Azure Red Hat OpenShift

25 /

4장
프로비저닝 전 – 엔터프라이즈 아키텍처에 관한 질문

35 /

5장
Azure Red Hat OpenShift 클러스터 프로비저닝

42 /

6장
프로비저닝 후 – 2일 차

59 /

7장
샘플 애플리케이션 배포

81 /

8장
애플리케이션 플랫폼 살펴보기

102 /

9장
다른 서비스와 통합

112 /

10장
워크로드 및 팀 온보딩

117 /

11장
결론

119 /

12장
용어집

1장

Microsoft 및 Red Hat과 상담하기

Azure Red Hat OpenShift에 관해 알아보고 계신다면 Microsoft와 Red Hat에 문의해 주십시오. 이 가이드에서 제공되는 Azure Red Hat Openshift 플랫폼 사용에 관한 유용한 지침 이외에도, Red Hat과 Microsoft는 이 가이드에서 다루지 않는 풍부한 리소스를 제공하여 사용 경험을 향상해 드릴 수 있습니다. 저희 두 회사는 협업을 통해 Azure Red Hat OpenShift가 고객님의 애플리케이션 혁신 요구 사항을 충족하는지 확인하고자 합니다.

Azure Red Hat OpenShift를 개발한 이유는 단순합니다. 바로 고객의 요청에 따른 것입니다. 그 어느 때보다 더 많은 양사의 공동 고객(대기업과 중소 기업)이 Red Hat의 포트폴리오를 Microsoft Azure로 배포하고 계십니다. 자체 관리형 오퍼링인 OpenShift on Azure는 몇 년 동안 완벽하게 지원되었지만 클러스터 관리 시설정, 배포, 2일 차 운영을 수행하려면 쿠버네티스 전문 지식과 관리할 시간이 필요합니다. 이로 인해 비즈니스 목표 달성에 매우 중요한 시간을 빼앗기게 됩니다.

점점 더 많은 고객이 관리형 오퍼링인 Azure Red Hat OpenShift로 성공적인 배포를 롤아웃함에 따라 설정 및 2일 차 운영에 소요되는 시간이 단축되고 애플리케이션에 더 오랜 시간을 집중하고 있습니다.

Red Hat의 핸즈온 경험을 기반으로 Azure Red Hat OpenShift에서 애플리케이션을 빌드하는 모범 사례를 고객에게 제공하기 위해 이 가이드를 작성했습니다. 이 가이드는 자습용으로 작성되었습니다. 도입부에서 결론까지 각 장을 차례대로 읽거나 필요한 특정 정보를 검색하실 수 있습니다.

이 가이드의 대상 독자:

이 가이드는 완전 관리형 Red Hat OpenShift 클러스터의 풀서비스 배포를 위해 Azure 및 Red Hat OpenShift를 사용함으로써 애플리케이션 빌드 및 배포 기능을 강화하고자 하는 개발자, 운영자, 플랫폼 아키텍트 등 IT 기술자를 대상으로 합니다.

이 가이드에서 다루는 내용

이 가이드에서는 조직 내 기술 검증(POC)에서 프로덕션 배포에 이르기까지 Azure Red Hat OpenShift 이해 및 도입을 위한 핵심 주제를 단계별로 설명해 드립니다.

2장: *Red Hat OpenShift 소개*에서는 먼저 Red Hat OpenShift를 소개하고 그토록 많은 개발자, 운영자, 플랫폼 아키텍트가 이 솔루션을 애플리케이션 플랫폼으로 선택하는 이유와 다양하게 활용하는 방법을 설명합니다. 그다음에 Red Hat OpenShift가 선호되는 플랫폼인 이유를 살펴봅니다.

3장: *Azure Red Hat OpenShift*에서는 관리형 서비스가 무엇이고 고객에게 어떤 방식으로 제공되는지 설명합니다.

4장: *프로비저닝 전 – 엔터프라이즈 아키텍처에 관한 질문*에서는 Azure Red Hat OpenShift를 배포하기 전에 고려할 중요한 질문에 대해 설명합니다. 여기에는 Azure Red Hat OpenShift를 실제로 배포한 많은 고객과 대화한 결과 얻게 된 다수의 모범 사례 지침이 포함되어 있습니다.

5장: *Azure Red Hat OpenShift 클러스터 프로비저닝*에서는 Azure Red Hat OpenShift 클러스터를 배포하는 데 필요한 주요 리소스가 있는 공식 도큐멘테이션 위치를 알려드립니다.

6장: *프로비저닝 전 – 2일 차*에서는 대개 "2일 차"라고 하는 프로비저닝 후 태스크에 관해 설명합니다. 가능한 경우 이 가이드는 관리형 서비스인 Azure Red Hat OpenShift를 이용하면 고객님이 직접 하는 것보다 더 수월해진다는 사실을 강조하고자 합니다.

7장: *Red Hat OpenShift에 애플리케이션 배포*는 이 플랫폼에 애플리케이션을 배포하는 방법에 관한 간략한 가이드입니다. 하지만 이러한 배포는 다른 환경에서 실행되는 Red Hat OpenShift와 전혀 다르지 않다는 사실에 주의해야 합니다.

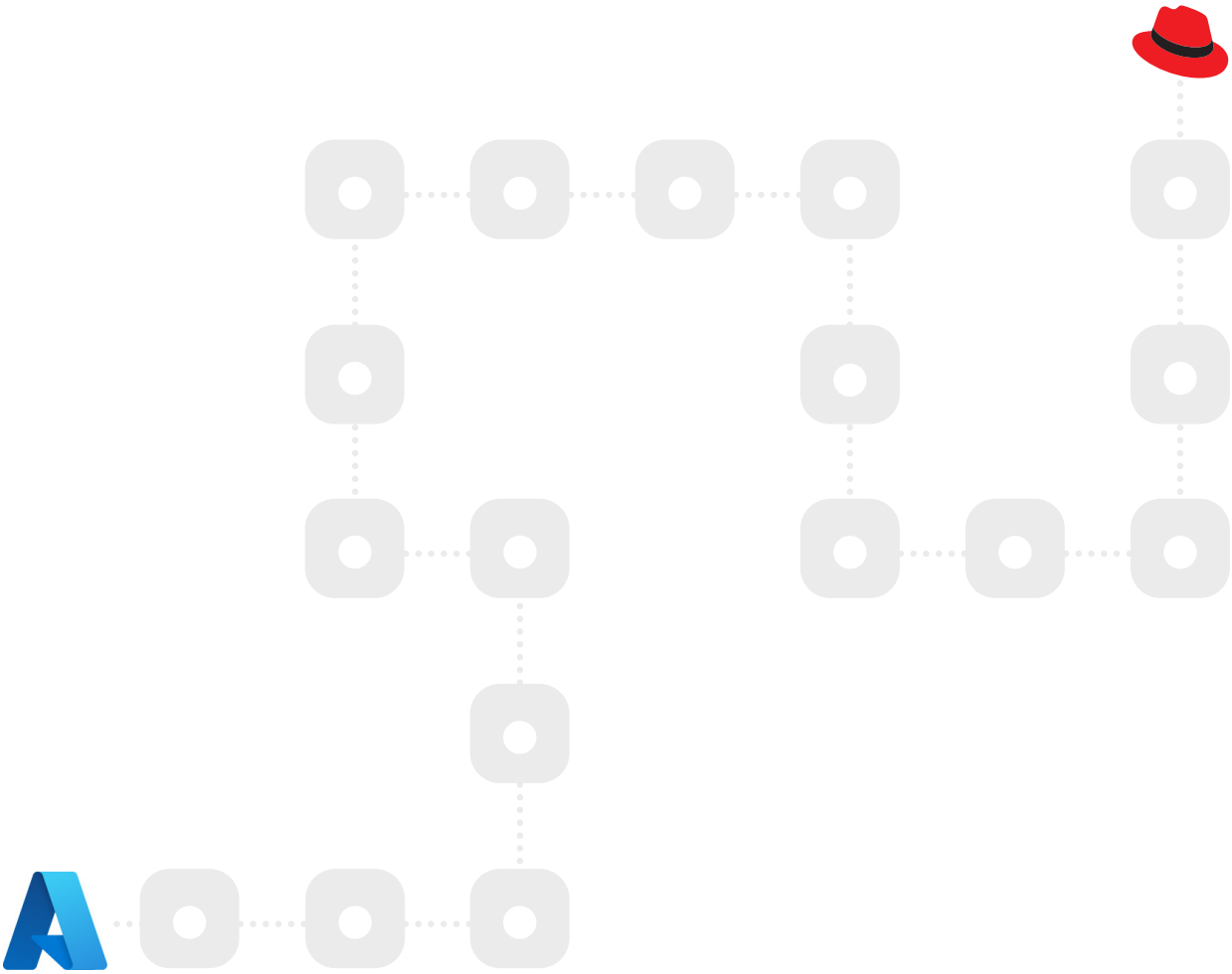
8장: *애플리케이션 플랫폼 살펴보기*에서는 컨테이너 레지스트리, 파이프라인, 서버리스, 이와 유사한 것 등 애플리케이션 플랫폼의 몇 가지 주요 기능에 대해 간략히 안내합니다.

9장: *다른 서비스와 통합*에서는 Azure Service Operator, Azure DevOps, 그리고 이와 유사한 서비스를 사용한 플랫폼 통합에 관해 설명합니다.

10장: *워크로드 및 팀 온보딩*에서는 애플리케이션 팀을 온보딩하고 조직 내에서 서비스를 도입하는 방법에 관한 몇 가지 모범 사례와 권장 사항을 제공하는 것으로 설명을 마칩니다.

11장: *결론*에는 맺는말이 포함되어 있습니다.

12장: *용어집*에서는 궁금한 용어를 찾아볼 수 있습니다.



2장

Red Hat OpenShift 소개

이 장에서는 Red Hat OpenShift의 정의, 사용 방법, 이 서비스로 Red Hat 고객이 일반적으로 체험한 이점에 대해 간략히 소개합니다. 이 장은 OpenShift를 처음 소개받는 경우 유용한 기본 지침의 역할을 합니다. 또는 이 플랫폼에 대해 좀 더 자세히 알아보고 싶은 경우 해당 정보를 빠르게 찾아볼 수 있는 보충 자료로 사용할 수 있습니다.

Red Hat OpenShift 개요

Red Hat OpenShift는 자동화된 풀스택 운영으로 하이브리드 클라우드 및 멀티클라우드 배포를 관리하는 엔터프라이즈 수준의 애플리케이션 플랫폼입니다. 개발자 생산성을 높이고 혁신을 촉진하도록 최적화되었습니다. Red Hat OpenShift가 지원하는 자동화된 운영과 간소화된 라이프사이클 관리를 통해 개발 팀은 새로운 애플리케이션을 빌드 및 배포하는 역량을 강화하고 운영 팀은 쿠버네티스 플랫폼을 프로비저닝, 관리, 확장할 수 있습니다.

개발 팀은 제공 프로세스 전반에서 보안 검사와 암호화 서명을 통해 수백 개 파트너의 검증된 이미지와 솔루션을 사용할 수 있습니다. 또한 개발 팀은 하나의 플랫폼에서 온디맨드 이미지와 광범위한 타사 클라우드 서비스를 바로 사용할 수 있습니다.

운영 팀은 빌트인 로깅 및 모니터링 기능으로 어디서나 팀 전체의 배포 상황을 파악할 수 있습니다. Red Hat 쿠버네티스 오퍼레이터에 포함된 고유한 애플리케이션 로직은 서비스가 구성될 뿐 아니라 최적의 성능을 제공하도록 조정되고 한 번의 터치로 간편하게 OS 업데이트 및 패치를 설치할 수 있도록 지원합니다. 간단히 말해 Red Hat OpenShift는 조직이 IT 및 개발 팀의 역량을 최대한 발휘할 수 있도록 지원하는 원스톱 솔루션입니다.

OpenShift 클라우드 서비스 – 비용 절감 및 비즈니스 이점

수천 곳 이상의 고객이 Red Hat OpenShift를 사용하여 애플리케이션 배포 방식을 변경하고, 고객과의 관계를 개선하며, 경쟁 우위를 확보해 업계 리더로 자리매김하고 있습니다. Red Hat OpenShift의 비즈니스 가치(2021년 3월)에 간략히 소개된 IDC의 연구 결과에 따르면, 인터뷰에 응한 조직들이 Red Hat OpenShift 플랫폼을 통해 비즈니스에 더 높은 품질과 더 시기적절한 애플리케이션과 기능을 제공하는 동시에 개발 및 IT 관련 비용과 직원 업무 시간 요구 사항을 최적화하여 많은 가치를 실현한 것을 알 수 있습니다.

주요 지표는 다음과 같습니다.

- 5년간 636%의 ROI(투자수익률)
- 10개월의 투자 회수 기간
- 5년간 운영 비용 54% 절감
- 매년 새로운 기능 3배 증가
- 애플리케이션 개발자 생산성 20% 향상
- 예상치 못한 다운타임 71% 감소
- IT 인프라 팀의 효율성 21% 상승

출처: IDC 백서, Red Hat 후원, *Red Hat OpenShift의 비즈니스 가치*, doc #US47539121, 2021년 3월

이러한 Red Hat OpenShift의 가치 외에도 Red Hat OpenShift 클라우드 서비스(Azure Red Hat OpenShift 포함)는 부가적인 비즈니스 이점을 제공합니다. The Total Economic Impact™ of Red Hat OpenShift Cloud Services - Cost Savings and Business Benefits라는 제목의 Forrester 연구에서는 몇 가지 재정적인 이점이 강조되었습니다.

OpenShift 클라우드 서비스의 주요 재정적 이점은 다음과 같습니다.

- 468%의 ROI(투자수익률)
- 408만 달러의 순현재가치(NPV)
- 6개월의 투자 회수 기간

이러한 재정적 이점 외에도 Forrester 리포트에서 제시하는 수량화된 이점의 주요 내용은 다음과 같습니다.

- **개발 속도 향상**: Red Hat OpenShift 클라우드 서비스를 통해 조직은 개발 사이클을 최대 70% 단축할 수 있습니다.
- **20%의 개발자 시간을 인프라 유지 관리 작업에서 재확보**: 인터뷰에 응한 사람들은 Red Hat OpenShift 클라우드 서비스로 인해 개발자가 애플리케이션 개발 인프라를 유지 관리할 필요가 없어 제품 또는 솔루션 구축에 온전히 집중할 수 있게 되었다고 답했습니다. 3년간 이렇게 개발자 시간을 재확보한 것은 230만 달러 이상의 가치가 있습니다.
- **운영 효율성 50% 향상**: 인터뷰에 응한 사람들은 Red Hat OpenShift 클라우드 서비스가 관리형 서비스이므로 이 솔루션을 사용하면서 이전에 인프라 관리를 담당하던 DevOps 직원 중 50%를 더 생산적인 다른 일에 재할당할 수 있게 되었다고 답했습니다. 3년 동안 이렇게 운영 효율성이 향상된 것은 130만 달러 이상의 가치가 있습니다.*

이 리포트는 다음과 같이 수량화되지 않은 이점도 발견했습니다.

- **개발자 만족 및 유지**: 인터뷰에 응한 사람들은 개발자가 Red Hat OpenShift 클라우드 서비스가 제공하는 이점을 통해 업데이트를 더 작은 크기로 분할하고, 매우 제한된 일정 내에 광범위한 테스트를 수행해야 하는 압박을 완화하고, 프로덕션 중에 긴급 사안에 대응해야 할 필요성을 줄일 수 있었다고 강조했습니다.
- **보안과 위험 감소**: 인터뷰에 응한 사람들은 Red Hat OpenShift 클라우드 서비스가 특정 기능 및 보안 업데이트를 자동화하여 수동 유지 관리의 필요성을 없애면서도 환경의 안전을 보장한 사례를 공유했습니다.
- **신뢰성**: 인터뷰에 응한 사람들은 Red Hat OpenShift 클라우드 서비스를 사용할 경우 환경이 확장되어도 운영 중단이나 시스템 장애가 덜 발생하므로 장기적으로 애플리케이션 플랫폼의 신뢰성이 더 높아졌다고 답했습니다.
- **이식성 및 비즈니스 연속성**: 인터뷰에 응한 사람들은 Red Hat OpenShift 클라우드 서비스를 통해 비즈니스 연속성이 보장되었고 이 서비스의 이식성, 확장성, 유연성으로 인해 재해 복구 전략에 도움이 되었다고 답했습니다.

출처: Forrester, *Red Hat OpenShift 클라우드 서비스의 Total Economic Impact*, 2021년 12월

***추가 자료:** 클라우드 서비스로 민첩성을 향상하는 기업들

"Red Hat OpenShift인가요, 아니면 기본 쿠버네티스인가요?" – 자체 쿠버네티스 애플리케이션 플랫폼 구축에 따른 비용

Red Hat OpenShift를 대개 "엔터프라이즈급 쿠버네티스"라고 하지만 무슨 뜻인지 바로 이해하기 어려울 수 있습니다. 고객들은 "OpenShift인가요, 아니면 기본 쿠버네티스인가요?"라고 자주 묻습니다. 하지만 **OpenShift는 이미 쿠버네티스를 사용하고 있다**는 것을 이해하는 것이 중요합니다. 전체 OpenShift 아키텍처에서 쿠버네티스는 OpenShift 플랫폼이 구축되는 기반과 OpenShift 실행을 위한 툴 대부분을 제공합니다.

쿠버네티스는 엄청나게 중요한 오픈소스 프로젝트로서, 클라우드 네이티브 컴퓨팅 재단의 핵심 프로젝트 중 하나이며 컨테이너 실행에 필수적인 기술입니다.

하지만 OpenShift의 잠재적 사용자가 실제 품을 수 있는 의문은 "**쿠버네티스만으로 애플리케이션을 실행할 수 있을까**?" 하는 것입니다. 많은 조직이 쿠버네티스를 배포하면서 엔터프라이즈 애플리케이션이라 하더라도 컨테이너를 단 며칠 만에 실행할 수 있음을 알게 됩니다. 하지만 2일 차 운영이 시작되고 보안 요구 사항이 증가해 더 많은 애플리케이션이 배포되고 나서야 쿠버네티스 기술로 자체 **서비스로서의 플랫폼(PaaS)**을 구축해야 하는 덫에 빠진 것을 깨닫는 조직이 있습니다. 이러한 조직은 오픈소스 인그레스 컨트롤러를 추가하고, 몇 가지 스크립트를 작성하여 자체 **지속적 통합/지속적 제공(CI/CD)** 파이프라인에 연결한 다음, 더 복잡한 애플리케이션을 배포하려고 하는데… 바로 이때 문제가 시작됩니다. 쿠버네티스를 배포하는 일이 수면 위로 드러난 빙산의 맨 윗 부분이라면 2일 차 관리의 복잡성은 배를 침몰시킬 정도로 방대한 수면 아래 숨겨진 부분이라고 할 수 있습니다.

이러한 어려움과 문제를 해결하는 데 착수할 수는 있지만 이러한 "쿠버네티스 기반 사용자 정의 PaaS"를 구축하고 유지 관리하려면 여러 명으로 구성된 운영 팀과 몇 주에서 몇 달까지의 시간이 필요한 경우가 많습니다. 이로 인해 조직의 효율성이 떨어지고, 보안 및 인증 관점에서 이를 지원하는 과정이 복잡해질 뿐 아니라 개발 팀을 온보딩할 때 모든 것을 처음부터 개발해야 하는 상황에 부닥치게 됩니다. 조직이 이 사용자 정의 쿠버네티스 플랫폼(즉 쿠버네티스) 구축 및 유지 관리와 관련된 모든 태스크를 성공적인 컨테이너 실행에 필요한 모든 추가 구성 요소와 함께 나열하고자 하는 경우 다음과 같이 그룹화할 수 있습니다.

- **클러스터 관리**: 여기에는 OS 설치, OS에 패치 적용, 쿠버네티스 설치, CNI 네트워킹 구성, 인증 통합, 인그레스 및 이그레스 설정, 퍼시스턴트 스토리지 설정, 노드 강화, 보안 패치 적용, 기본 클라우드/멀티클라우드 구성이 포함됩니다.
- **애플리케이션 서비스**: 여기에는 로그 집계, 상태 점검, 성능 모니터링, 보안 패치 적용, 컨테이너 레지스트리, 애플리케이션 스테이징 프로세스 설정이 포함됩니다.
- **개발자 통합**: 여기에는 CI/CD 통합, 개발자 툴/IDE 통합, 프레임워크 통합, 미들웨어 호환성, 애플리케이션 성능 대시보드 제공, RBAC가 포함됩니다.

목록에 추가할 수 있는 작업과 기술이 더 많이 있지만(이러한 활동 중 대부분은 조직이 컨테이너를 사용하는 데 필수적임) 이 모든 것을 설정하는 과정에서 겪는 복잡성과 소요되는 시간, 노력은 각각의 요소를 지속적으로 유지 관리하는 것에 비하면 대수롭지 않은 것입니다. 통합할 때마다 철저히 테스트해야 하고, 각 구성 요소 및 활동은 출시 주기, 보안 정책, 패치가 서로 달라야 합니다.

기본 쿠버네티스와 달리 Red Hat OpenShift를 통해 얻을 수 있는 장점

조직이 프로덕션 단계에서 기본 쿠버네티스에 구축된 컨테이너를 실행하게 되면 앞 섹션에서 언급한 구성 요소들은 일반적인 방식으로 설치되고 서로 통합되어 자체 설계에 따른 일종의 애플리케이션 플랫폼을 구성하게 됩니다.

Azure Red Hat OpenShift는 앞 섹션에서 언급한 모든 구성 요소를 하나의 플랫폼으로 통합하므로, IT 팀은 손쉽게 운영할 수 있고 애플리케이션 팀은 태스크 실행에 필요한 모든 것을 얻을 수 있습니다. 이와 관련된 모든 주제는 본 가이드 후반부에서 더 자세히 다루겠지만, 이를 염두에 두고 둘 사이의 핵심적인 차이점에 대해 살펴보겠습니다.

- **배포 용이성**: 쿠버네티스에서 애플리케이션을 배포하는 데 시간이 오래 걸릴 수 있습니다. 이러한 배포에는 GitHub 코드를 머신으로 풀링하고, 컨테이너를 가동하여 Docker Hub와 같은 레지스트리에서 호스팅하고, 끝으로 CI/CD 파이프라인을 이해하는 작업이 수반되며, 이 과정은 매우 복잡할 수 있습니다. 반면, OpenShift는 이처럼 힘든 일과 백엔드 작업을 자동화하므로 프로젝트를 생성하고 코드를 업로드하기만 하면 됩니다.
- **보안**: 오늘날 대부분의 쿠버네티스 프로젝트는 여러 명의 개발자 및 오퍼레이터로 구성된 팀에서 수행되고 있습니다. 현재 쿠버네티스가 RBAC, IAM과 같은 제어 기능을 지원하고 있기는 하지만 여전히 시간이 소요되는 수동 설정 및 구성 작업이 필요합니다. Red Hat과 OpenShift는 수년에 걸친 경험 끝에 고객이 즉시 활용할 수 있는 보안 모범 사례를 식별하는 큰 성과를 거두었습니다. 새로운 사용자를 추가하면 OpenShift가 네임스페이스 지정, 다양한 보안 정책 생성과 같은 작업을 처리합니다.
- **유연성**: Azure Red Hat OpenShift 사용 시 배포, 관리 및 업데이트와 관련해 잘 알려진 모범 사례를 이용할 수 있습니다. 백엔드 내에서 발생하는 모든 힘든 일은 자동으로 처리되므로 수동 작업이 별로 필요 없고 애플리케이션을 더 빨리 제공할 수 있습니다. 쿠버네티스 플랫폼을 활용하면 간소화된 접근 방식을 통해 작업을 완료하고 이익을 얻는 방법에 대해 관심이 있는 팀에게 효과가 있을 뿐 아니라 프로세스를 개발할 때 더 많은 유연성과 창의성을 제공하는 CI/CD DevOps 파이프라인을 수동으로 사용자 정의할 수 있습니다.

- **일상적 운영**: 클러스터는 여러 VM이 모인 그룹으로 구성되므로 운영 팀은 클러스터에 추가해야 하는 새로운 VM을 가동하지 않을 수 없게 됩니다. 쿠버네티스를 통한 구성 프로세스는 시간이 오래 걸리고 복잡할 수 있으므로 셀프 등록 또는 클라우드 자동화 등을 설정하려면 스크립트를 개발해야 합니다. Azure Red Hat OpenShift를 사용하면 클러스터 프로비저닝, 확장 및 업그레이드 작업이 이 플랫폼에 의해 자동화되고 관리됩니다.
- **관리**: 모든 배포판과 함께 제공되는 쿠버네티스 표준 툴 및 대시보드를 이용할 수 있지만 대부분의 개발자는 더 완벽하고 강력한 플랫폼이 필요합니다. Azure Red Hat OpenShift는 쿠버네티스 API를 기반으로 구축되는 뛰어난 웹 콘솔과 운영 팀이 자체 워크로드를 관리할 수 있는 기능을 제공합니다.

*8장: 애플리케이션 플랫폼 살펴보기*에는 Azure Red Hat OpenShift가 제공하는 다수의 중요 부가 가치 기능에 대해 안내하는 설명이 포함되어 있습니다.

요약

다수의 Red Hat 및 Microsoft 공동 고객은 컨테이너화된 애플리케이션을 도입하는 데 사용할 애플리케이션 플랫폼으로 Azure Red Hat OpenShift를 선호합니다.

다음 장에서는 클라우드 서비스인 Red Hat OpenShift에 대해 살펴보면서 이 서비스의 아키텍처, 통합 및 관리에 대해 자세히 알아보겠습니다.

3장

Azure Red Hat OpenShift

Azure Red Hat OpenShift를 사용하면 실행의 기반이 될 인프라의 구축 및 관리에 대한 걱정 없이 완전 관리형 Red Hat OpenShift 클러스터를 배포할 수 있습니다.

Azure Red Hat OpenShift는 Red Hat과 Microsoft가 공동으로 엔지니어링, 운영, 지원하는 클라우드 네이티브 온디맨드 애플리케이션 플랫폼입니다. 전문화된 **사이트 신뢰성 엔지니어링**(**SRE**) 팀은 OpenShift 클러스터를 자동화, 확장, 보호하고 상호 협력하여 통합 지원 경험을 제공합니다. Azure Red Hat OpenShift를 사용하면 운영할 가상 머신이 없으므로 패치를 적용할 필요가 없습니다. 컨트롤 플레인 노드와 작업자 노드의 경우 Red Hat과 Microsoft가 사용자를 대신하여 패치 적용, 업데이트, 모니터링을 수행합니다. Azure Red Hat OpenShift 클러스터는 Azure 서브스크립션으로 배포되고 Azure 청구서에 포함됩니다.

Azure Red Hat OpenShift로 더 빠르게 애플리케이션 제공:

- 개발자가 빌트인 CI/CD 파이프라인을 통해 혁신하고, MySQL, PostgreSQL, Redis, Azure Cosmos DB 등 수백 가지 Azure 서비스에 애플리케이션을 손쉽게 연결할 수 있도록 지원
- 자동화된 프로비저닝, 구성 및 운영으로 복잡성을 줄이고 생산성 저해 요인을 제거
- 몇 분 안에 3개의 작업자 노드가 있는 고가용성 클러스터를 시작한 후 메모리/CPU 사용량이 많은 표준 작업자 노드를 선택하여 애플리케이션 수요 변화에 따라 확장할 수 있는 맞춤형 확장성
- 통합된 지원 경험을 통한 엔터프라이즈 수준의 운영, 보안 및 컴플라이언스

이제 Red Hat OpenShift가 구축되고 운영되는 방식 이면의 기술 세부 정보에 대해 자세히 알아보겠습니다.

아키텍처

Azure Red Hat OpenShift는 Azure 인프라 서비스인 가상 머신, 네트워크 보안 그룹, 스토리지 계정, 기타 Azure 서비스를 Red Hat OpenShift 설치의 기반으로 사용합니다. Azure 아키텍처는 Azure 서브스크립션으로 완전히 배포되므로 계정 내에서 이미 실행 중인 다른 서비스와 손쉽게 통합할 수 있습니다.

OpenShift 자체는 함께 작동하는 더 작고 분리된 단위들로 구성된 마이크로서비스 기반 아키텍처를 호스팅하는 Red Hat Enterprise Linux CoreOS 위에 구축됩니다. 각 가상 머신에서는 쿠버네티스 클러스터의 기반인 `kubelet` 서비스가 실행 중입니다. 컨트롤 플레인에서 실행 중인 이 아키텍처 이면의 데이터베이스는 신뢰할 수 있는 클러스터링된 키-값 저장소인 **etcd**입니다.

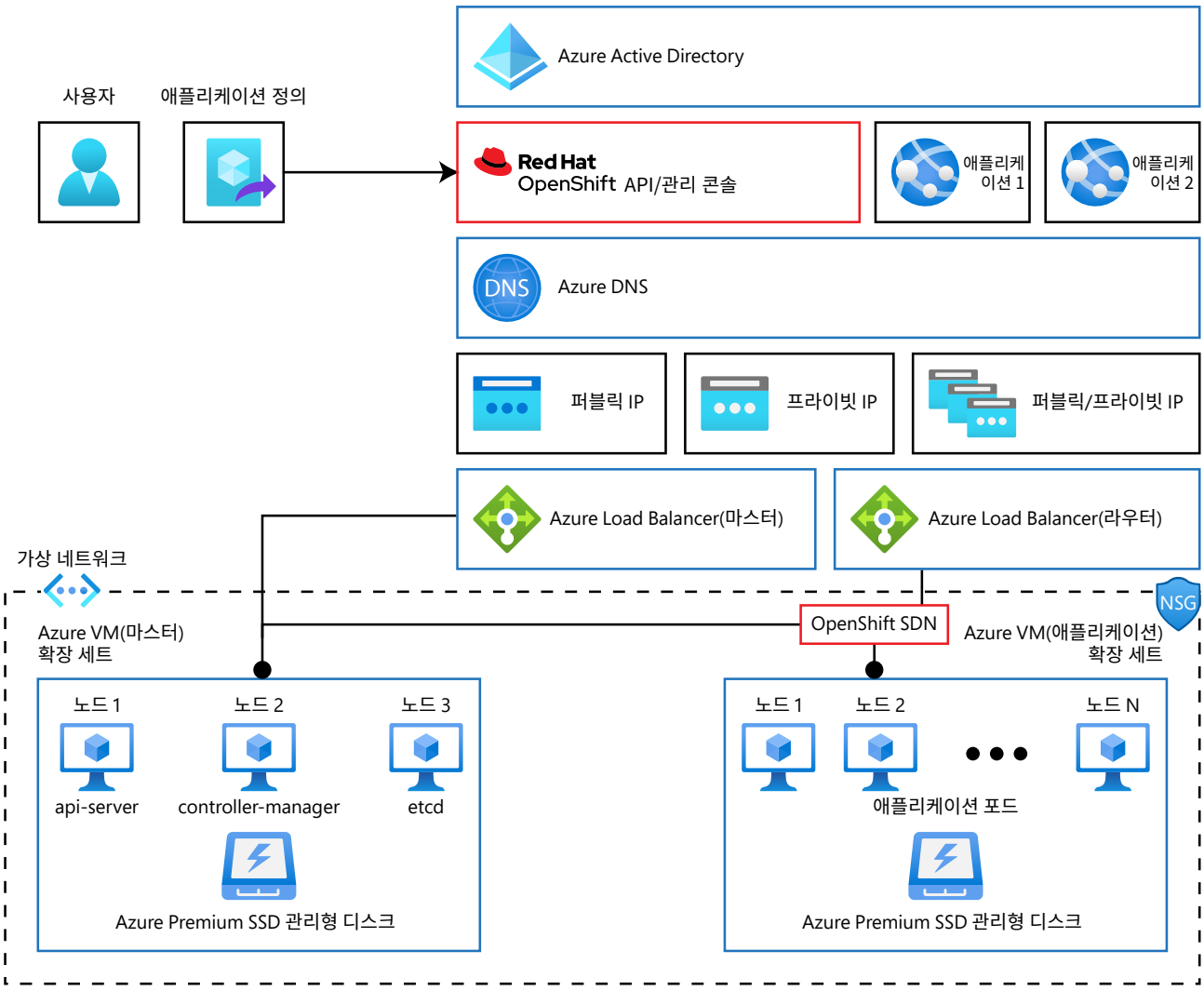


그림 3.1: Azure Red Hat OpenShift 아키텍처

다음 두 섹션(*컴퓨팅 – 제어 및 작업자 노드* 및 *네트워킹*)에서는 이 다이어그램에 포함된 것에 대해 자세히 알아보겠습니다.

컴퓨팅 – 제어 및 작업자 노드

Red Hat OpenShift의 아키텍처는 세 가지 특정 역할 중 하나(제어, 인프라 또는 애플리케이션)를 담당하는 가상 머신(노드)에서 실행됩니다. 하지만 Azure Red Hat OpenShift 버전 4는 작성 시점에 인프라 노드를 지원하지 않으므로 이 책에서는 제어 및 작업자 노드만 다룹니다.

제어 노드(이전 명칭은 **마스터** 노드)는 쿠버네티스 클러스터에 필수적인 구성 요소가 포함된 가상 머신입니다. 이러한 구성 요소로 API 서버, 컨트롤러 관리자 서버, 스케줄러, etcd 등을 들 수 있습니다. 이 구성 요소들은 쿠버네티스 클러스터를 관리하고 작업자 노드에서 실행할 포드를 예약합니다.

- **쿠버네티스 API 서버** 포드, 서비스, 복제 컨트롤러용 데이터를 검증하고 구성합니다. 또한 클러스터의 공유 상태에 초점을 제공합니다.
- **쿠버네티스 컨트롤러 관리자** etcd를 모니터링하여 복제, 네임스페이스, 서비스 계정 컨트롤러 오브젝트 등의 오브젝트에 변경 사항이 있는지 확인한 다음, API를 사용해 지정된 상태를 적용합니다. 이러한 몇 가지 프로세스는 하나의 능동적 리더로 한 번에 클러스터를 생성합니다.
- **쿠버네티스 스케줄러** 지정된 노드가 없는 새로 생성된 포드를 감시하고 이 포드를 호스팅하기에 가장 좋은 노드를 선택합니다.
- **etcd** 영구 마스터 상태를 저장하고, 다른 구성 요소는 etcd의 변경 사항을 감시하여 자신을 지정된 상태로 전환합니다.

애플리케이션 노드는 애플리케이션이 실제로 실행될 곳입니다.

쿠버네티스 클러스터의 각 노드는 `kubelet`이라는 서비스를 실행합니다. 이 서비스는 **컨테이너 런타임 인터페이스**(**cri-o**), 서비스 프록시, 그리고 각 노드에서 실행되는 기타 필수 서비스를 유지 관리합니다. 모든 노드는 OpenShift의 소프트웨어 정의 네트워킹 기술과 연결되어 있습니다. Azure Red Hat OpenShift에서 이 기술은 Azure 가상 네트워크 위에서 실행되는 **OVN(Open Virtual Networking)**입니다.

Azure Red Hat OpenShift는 스토리지용 Azure 디스크에 연결된 Azure 가상 머신에서 실행되는 노드를 생성합니다. 1TB의 꽤 큰 디스크임을 알 수 있습니다. 이는 Azure에서 스토리지 성능에 대한 IOPS 보장이 디스크의 크기와 연계되어 있기 때문입니다. 1TB의 크기는 etcd 데이터베이스가 사용하는 기본 디스크에 충분한 대역폭을 보장합니다.

네트워킹

Azure Red Hat OpenShift는 서브넷이 컨트롤 플레인 노드와 작업자 노드에 각기 한 개씩 구성된 단일 Azure 가상 네트워크가 필요합니다. 두 네트워크의 최소 크기는 /27(32개의 주소)입니다. 하지만 나중에 클러스터를 확장할 생각이 없다면 이 크기를 너무 작게 설정하지 않도록 주의하세요.

두 개의 Azure Load Balancer(다이어그램에서 각기 **마스터**와 **라우터**로 레이블이 지정되어 있음)는 트래픽을 다음과 같이 유도합니다.

- 마스터/컨트롤 플레인 로드 밸런서: OpenShift API의 사용자에게 인그레스 트래픽을 전송합니다. 또한 컨트롤 플레인 노드에 아웃바운드 연결을 제공합니다.
- 라우터/애플리케이션 로드 밸런서: OpenShift에서 실행되는 애플리케이션으로 OpenShift "라우터" 또는 인그레스 컨트롤러를 통해 인그레스 트래픽을 전송합니다. 또한 작업자 노드에 아웃바운드 연결을 제공합니다.

An excellent article about the 네트워킹 구성, 요구 사항 및 제한 사항에 관한 탁월한 문서를 [네트워킹 도큐멘테이션](#)에서 보실 수 있습니다.

다른 Azure 서비스와 통합

Azure 기반 네이티브 서비스인 Azure Red Hat OpenShift를 고객님이 Azure에서 사용하는 데 익숙한 다수의 서비스와 함께 배포하고 통합할 수 있습니다. 다음은 일반적인 통합이 존재하는 Azure 인프라 서비스 중 일부에 대해서만 간략히 살펴본 내용입니다.

*그림 3.2*는 OpenShift on Azure의 여러 가지 일반적인 Azure 서비스 통합 포인트를 보여줍니다.

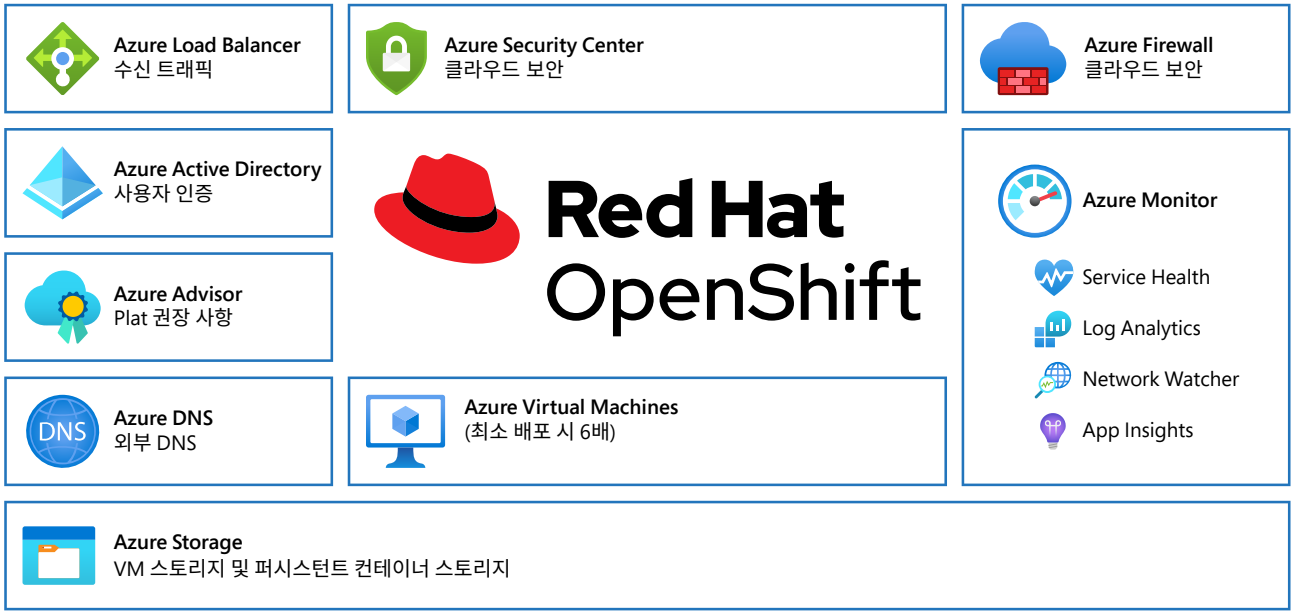


그림 3.2: 일반적인 Azure 서비스 통합 포인트

또한 OpenShift에서 실행되는 애플리케이션은 Azure Service Operator를 사용해 Azure 서비스와 매우 긴밀하게 통합될 수 있습니다. 이에 관해서는 나중에 *9장: 다른 서비스와 통합*에서 더 자세히 다루겠습니다.

관리

Azure Red Hat OpenShift 서비스의 일부로 관리되는 것이 무엇인지 이해한다는 것은 간단히 말해 클러스터 오퍼레이터에 이르기까지 데이터센터의 모든 것을 "관리되는" 것으로 간주하는 것입니다. 클러스터 오퍼레이터는 컨트롤 플레인 노드에서 실행되는 서비스이며 클러스터 모니터링, 업데이트, 상태에 주의를 기울입니다. 이는 Microsoft와 Red Hat이 이러한 구성 요소를 공동으로 모니터링, 유지, 관리하여 클러스터를 항상 온라인 상태로 유지함을 뜻합니다. 클러스터 오퍼레이터 위에 있는 모든 것은 계속해서 고객의 책임입니다.

Azure Red Hat OpenShift 소비자인 고객님께 클러스터에 대한 완전한 **클러스터 관리자** 액세스 권한이 부여되므로 클러스터를 여러 부분으로 분할하지 않을 책임을 고객님도 일부 공유하게 됩니다. 클러스터로 할 수 있는 것과 없는 것이 무엇인지 알려면 지원 정책을 이해하는 것이 중요합니다.

Microsoft와 Red Hat이 클라우드 서비스의 일부로 수행할 작업이 정확히 무엇인지에 관한 자세한 설명은 Azure Red Hat OpenShift 책임 매트릭스에 기술되어 있습니다.

인증 및 권한 부여

Azure Active Directorysms Azure Red Hat OpenShift 클러스터에 인증을 제공하는 일반적인 방법입니다. 하지만 필수는 아니며 Login with GitHub 또는 간단한 "비밀번호 파일"과 같은 다른 인증 메커니즘을 대신 사용할 수 있습니다.

Azure Active Directory가 사용되는 경우 Azure Red Hat OpenShift와 쿠버네티스 API는 인증 요청을 전달합니다. 사용자는 자신의 자격 증명을 제시하고 맡은 역할에 따라 권한을 부여받습니다.

인증 계층은 Azure Red Hat OpenShift API에 대한 요청과 연결된 사용자를 식별합니다. 그런 다음 권한 부여 계층은 요청하는 사용자에 관한 정보를 사용하여 해당 요청을 허용해야 하는지 여부를 결정합니다.

Azure Active Directory 구성에 관한 지침은 *인증 - Azure Active Directory* 섹션에 기술되어 있습니다. 이 프로세스는 Azure Active Directory 대신에 다른 인증 공급자가 사용되는 경우 기능적으로 매우 유사합니다.

권한 부여는 Azure Red Hat OpenShift 정책 엔진에서 처리됩니다. 이 엔진은 "포드 생성" 또는 "서비스 나열"과 같은 작업을 정의하고 이러한 작업을 정책 문서에서 역할로 그룹화합니다. 역할은 사용자 또는 그룹 식별자에 의해 사용자 또는 그룹에 바인딩됩니다. 사용자 또는 서비스 계정이 작업을 시도하면 이 정책 엔진은 계속 진행하도록 허용하기 전에 사용자(예: 고객 관리자 또는 현재 프로젝트의 관리자)에게 할당된 한 개 이상의 역할을 점검합니다.

클러스터 역할, 로컬 역할, 클러스터 역할 바인딩, 로컬 역할 바인딩, 사용자, 그룹, 서비스 계정 간의 관계는 다음과 같이 *그림 3.3*에 정리되어 있습니다.

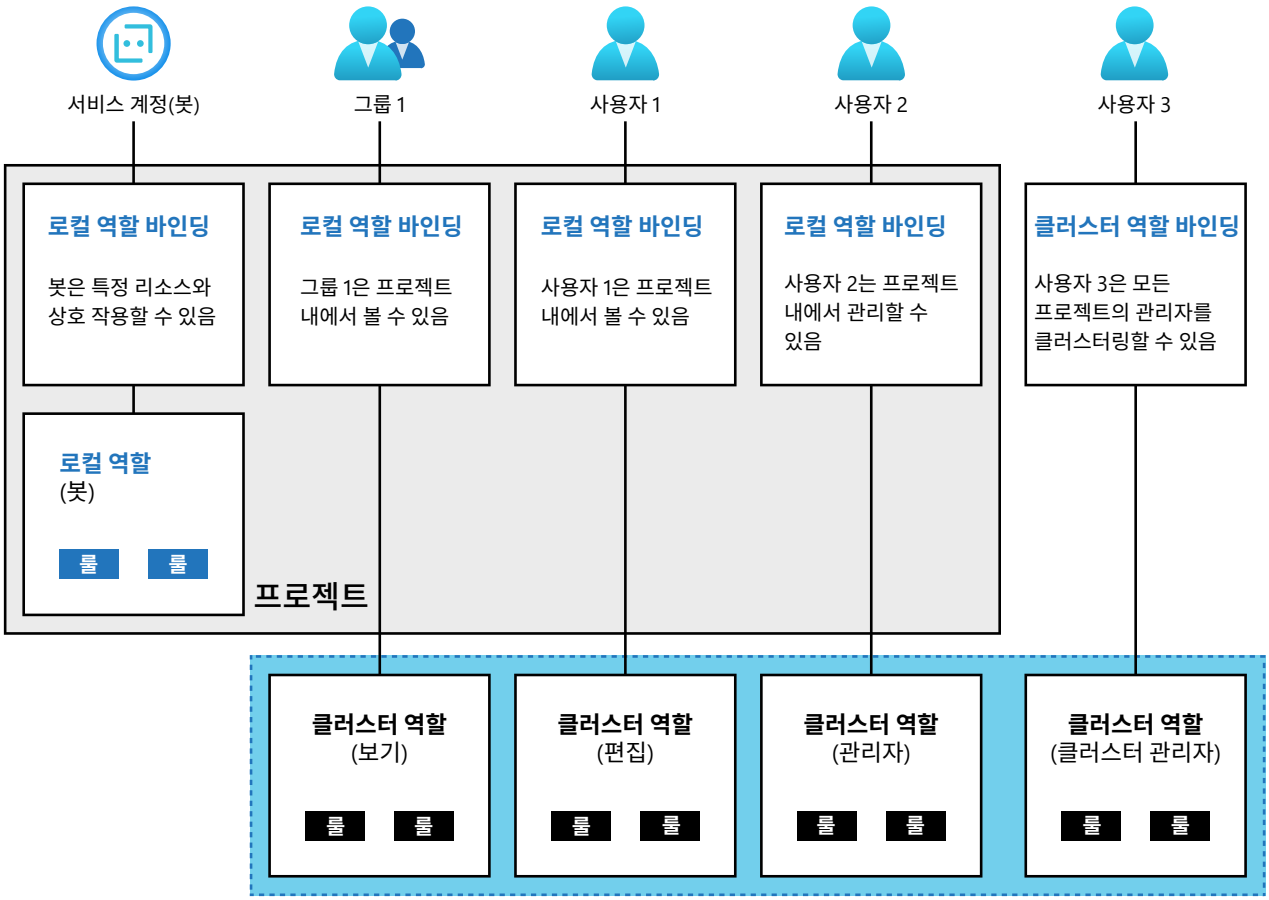


그림 3.3: 역할 간의 관계

Red Hat OpenShift 도큐멘테이션에는 인증 공급자 전체 목록이 있습니다.

지원

Azure Red Hat OpenShift는 지원을 관리하는 방식이 독특합니다. Microsoft와 Red Hat 지원 팀은 글로벌 [사이트 신뢰성 엔지니어링(SRE)] 팀과 협력하여 서비스를 더 쉽게 운영할 수 있도록 지원합니다.

고객은 Azure Portal에서 지원을 요청하고 이러한 요청이 Azure 플랫폼 수준이든 OpenShift 수준이든 관계없이 신속하게 처리하기 위해 Microsoft 및 Red Hat 엔지니어가 요청을 분류하고 처리합니다.

통합 지원 프로세스의 개요는 다음과 같습니다.

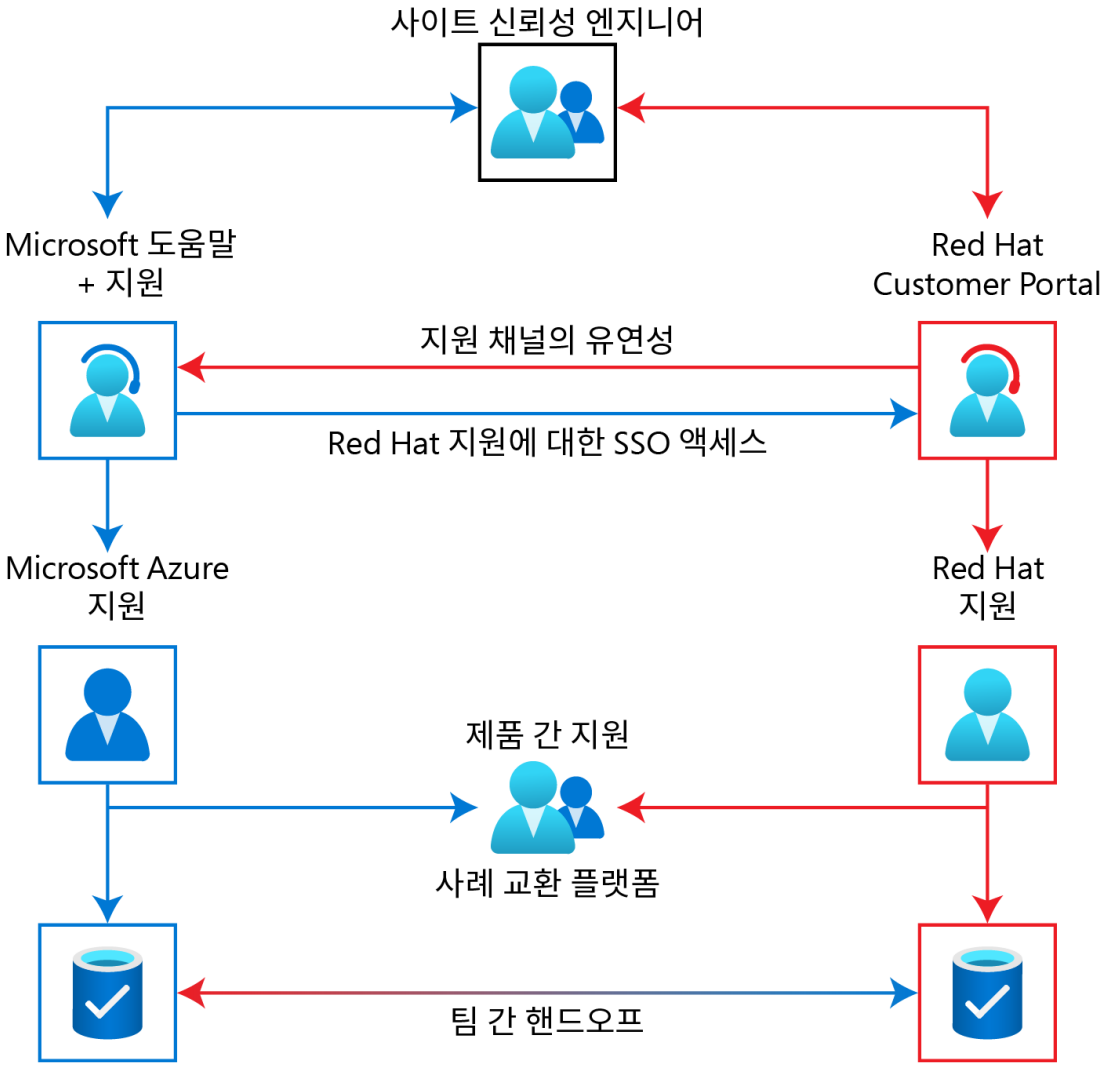


그림 3.4: 통합 지원 프로세스

*그림 3.4*는 고객이 Microsoft 지원 포털 또는 Red Hat 지원 포털에서 지원 티켓을 제출할 수 있음을 보여줍니다. Red Hat 지원 포털에 액세스하려면 클러스터를 OpenShift Cluster Manager에 등록해야 한다는 점에 유의하세요.

Microsoft의 지원 엔지니어는 사례 교환 플랫폼을 통해 고객의 동의를 받아 어느 단계에서든 Red Hat과 원활하게 협업할 수 있습니다. 이는 Microsoft와의 협업을 요청할 때 Red Hat 지원 엔지니어에게도 마찬가지로 적용됩니다. 양사의 지원 엔지니어는 SRE 팀에 액세스할 수 있고, SRE 팀은 필요한 경우 클러스터 복구를 위해 문제 해결 조치를 취할 수 있습니다.

가격책정 및 서브스크립션

Azure Red Hat OpenShift가 DIY(Do-It-Yourself) 설치보다 더 나은 주요 이점 중 하나는 컴퓨팅, 네트워크, 스토리지, Azure 인프라뿐 아니라 OpenShift 서브스크립션도 별도로 청구되는 대신 Azure 서브스크립션을 통해 모두 청구된다는 것입니다. 솔루션에 드는 비용이 대략 어느 정도인지 알아보려면 Azure 가격 책정 계산기에서 견적 빌더에 Azure Red Hat OpenShift를 추가하기만 하면 됩니다.

다음은 가격 책정 페이지에 대한 이해를 돕기 위한 가이드입니다.

1. 검색창에 *openshift*를 입력하면 표시되는 OpenShift를 새로운 추정치에 추가합니다.

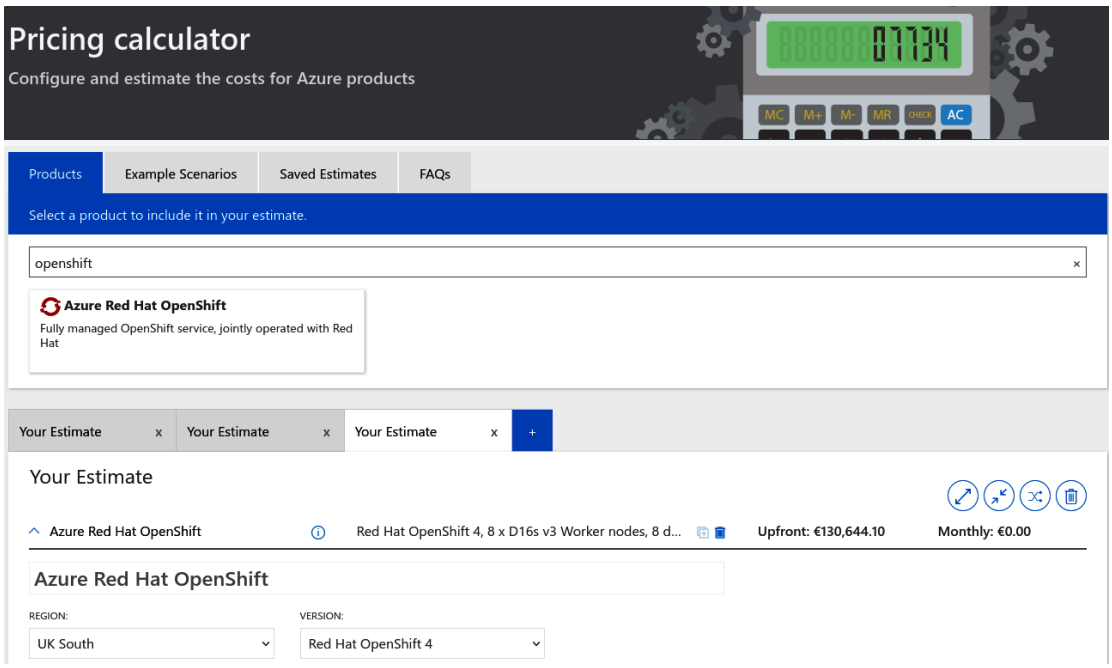


그림 3.5: 가격 책정 계산기 패널

2. 페이지 하단에 가격 단위를 현지 통화로 변경할 수 있는 드롭다운 콤보 상자가 표시됩니다. 이 예시에서는 영국 파운드화(£)가 사용됩니다.
3. Azure Red Hat OpenShift를 배포할 Azure 지역을 설정합니다. Azure 지역 전반의 다양한 컴퓨팅 비용을 감안해 지역에 따라 가격이 조금씩 변동됩니다.

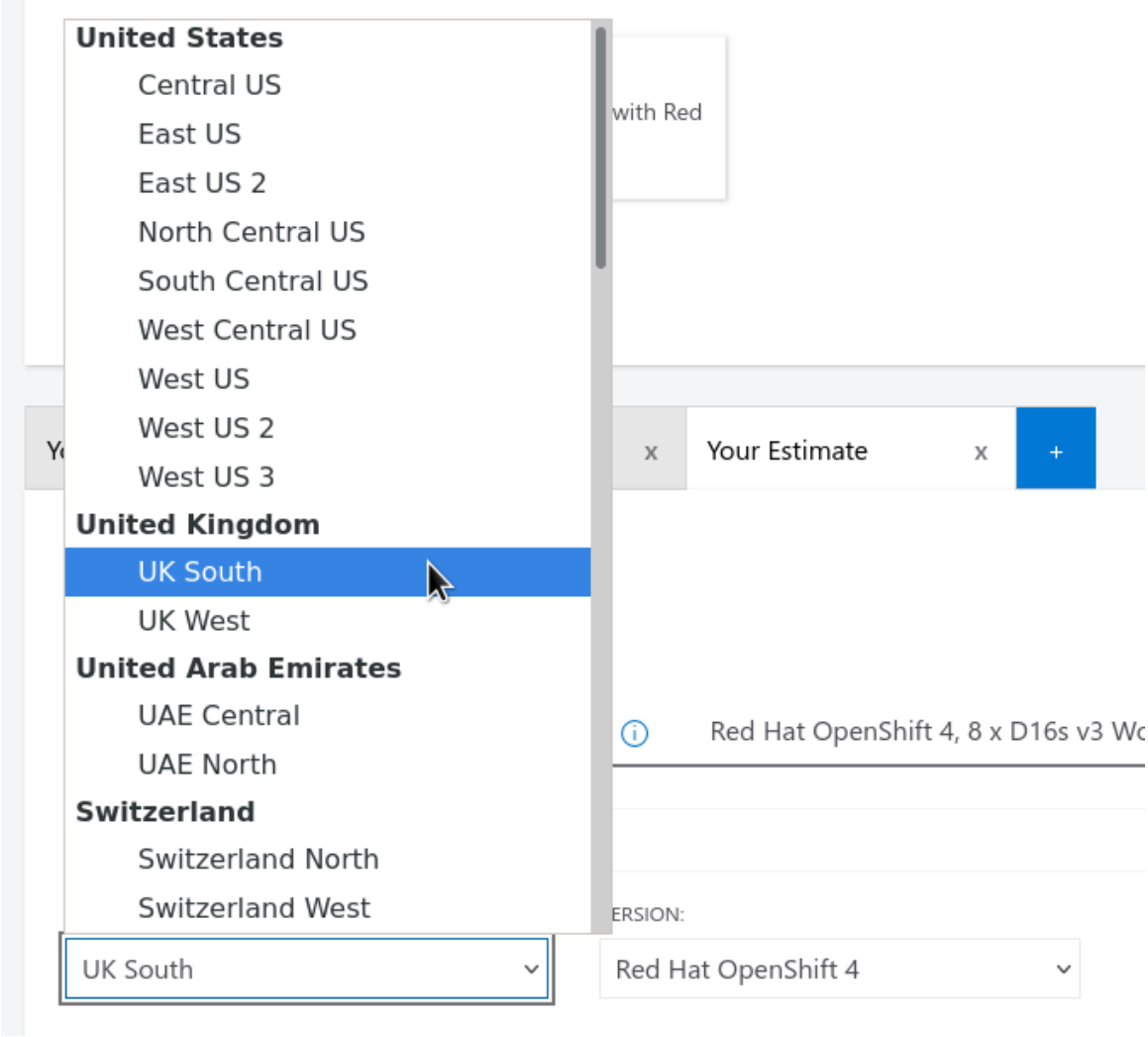


그림 3.6: Azure 지역 선택

4. **작업자 노드**는 실제 애플리케이션이 실행될 곳입니다. **절약 옵션**에는 두 섹션이 있습니다. **라이선스**는 OpenShift 서브스크립션의 비용을 가리키는 반면, **가상 머신** 섹션은 클러스터 실행에 필요한 컴퓨팅 서비스의 비용을 가리킵니다. 클러스터에서 지원되는 작업자 노드의 최대 수는 100개입니다.

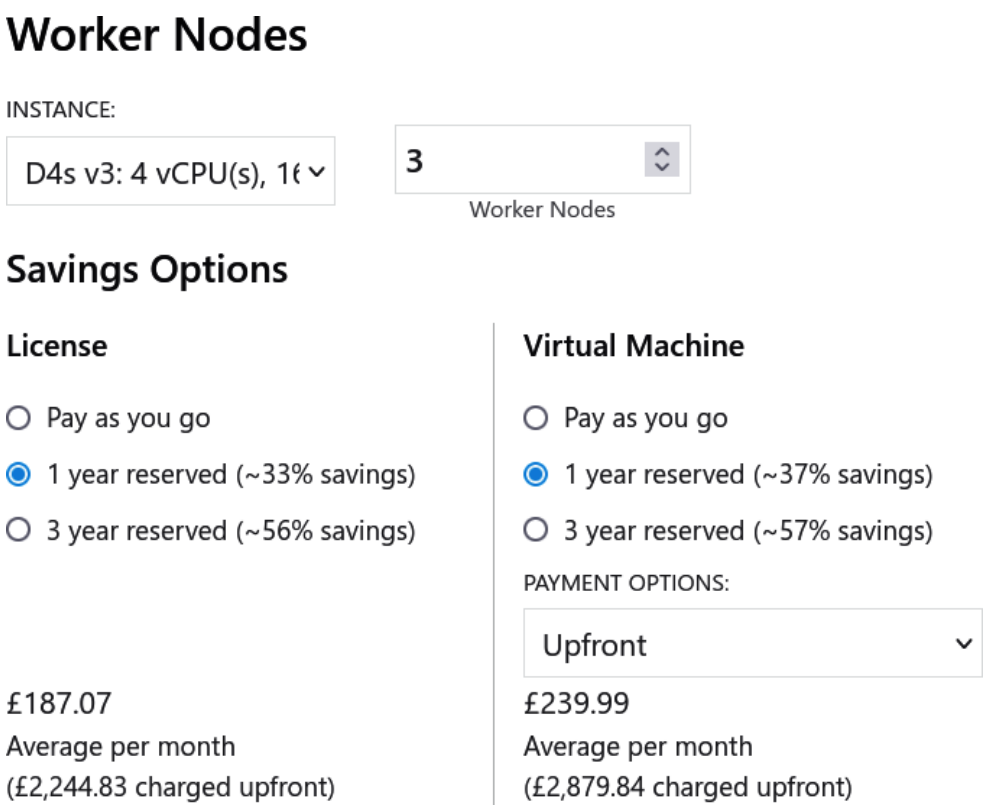


그림 3.7: 작업자 노드 세부 정보

5. **관리형 OS 디스크**는 운영 체제 파티션을 위해 Red Hat CoreOS가 사용하는 디스크의 크기를 가리킵니다. 이것은 별도로 프로비저닝되는 애플리케이션 퍼시스턴트 볼륨이 사용하는 스토리지를 가리키는 것이 아닙니다.

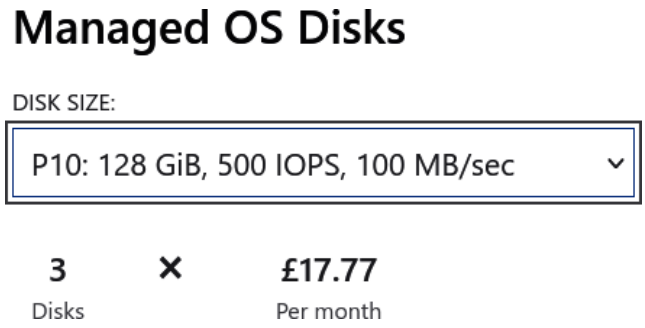


그림 3.8: 관리형 OS 디스크

6. **마스터 노드**라는 유사한 섹션이 있습니다. 마스터 노드는 현대적인 용어로는 일반적으로 '제어 노드'라고 합니다. 제어 노드에는 작업 노드에 비해 훨씬 더 큰 디스크가 연결되어 있다는 점에 유의하세요. 이는 Azure에서 더 큰 디스크 크기로 제공되는 더 높은 IOPS 및 처리량이 필요하기 때문입니다. 컨트롤 플레인 노드의 정확한 개수는 클러스터 안정성을 위해 항상 3개이어야 합니다.

계산기는 참고 가격을 알려주는 용도로 설계되었고, 실제 가격은 사용량에 따라 변동됨에 유의하세요.

Azure 예약 가상 머신 인스턴스

예약 인스턴스는 일정 기간, 즉 1~3년간 해당 리소스를 소비하겠다는 약속을 가리킵니다. Azure Red Hat OpenShift 가상 머신용 예약 인스턴스 옵션이 있습니다.

클러스터가 더 오랜 기간 동안 실행될 것을 알고 있는 조직은 IaaS 할인 폭을 늘리기 위해 일반적으로 예약 인스턴스를 선택합니다. 예를 들어 프로덕션 환경은 예약 인스턴스를 사용해 실행되는 경우가 많습니다. 예약 인스턴스는 클러스터의 서비스 수준 또는 아키텍처를 변경하지 않는다는 것을 알아두면 도움이 됩니다.

Azure 예약 인스턴스 자세히 알아보기

요약

이 장에서는 Azure Red Hat OpenShift 관리형 클라우드 서비스에 관해 자세히 알아보았습니다. 이 서비스의 아키텍처를 간략히 살펴본 후 다른 Azure 서비스와의 통합(*9장: 다른 서비스와 통합*에서 더 자세히 다룰 예정임)에 대해 관리, 인증, 지원, 가격 책정 관련 고려 사항과 함께 알아보았습니다.

다음 장에서는 Azure Red Hat OpenShift 배포 중 프로비저닝 전 단계에서 조직이 답해야 할 질문과 의사 결정 사안에 대해 중점적으로 알아보겠습니다.

4장

프로비저닝 전 – 엔터프라이즈 아키텍처에 관한 질문

많은 조직이 Azure Portal에서 Azure Red Hat OpenShift를 살펴볼 수 있습니다. 그리고 도큐멘테이션을 통독하면 약간의 노력만으로도 Red Hat OpenShift로 클러스터를 성공적으로 배포할 수 있습니다. 하지만 조금만 더 시간을 들여 배포를 계획하고 미리 몇 가지 질문을 던진다면 나중에 클러스터를 삭제하고 다시 프로비저닝하느라 소요될 시간을 훨씬 더 절약할 수 있습니다.

이 장은 다수의 고객과 협력하는 과정에서 쌓은 실제 핸즈온 경험에 기반을 두고 있습니다. 이 장에서는 프로비저닝 전에 먼저 해결해야 하는 다수의 일반적인 질문에 대해 다룹니다. 이 장에서 다루는 내용은 다음과 같습니다.

- 스테이징, 프로덕션 등을 포함해 몇 개의 클러스터가 필요하고 서로 유사한가
- 퍼블릭 및 프라이빗 네트워크 가시성
- 온프레미스 솔루션에 대한 연결성과 같은 하이브리드 연결성

필요한 클러스터 개수를 산출하는 방법부터 알아보겠습니다.

몇 개의 클러스터가 필요한가?

여러 가지 OpenShift 배포 패턴이 있지만 공통된 질문은 "조직에 필요한 클러스터는 몇 개인가?"입니다. 물론 이는 조직이 어떻게 결정하느냐에 달려 있는 사안이지만 다음 문단에 있는 지침이 클러스터 개수를 산출하는 데 도움이 될 것입니다.

라이프사이클 단계: 개발, 테스트, 프로덕션

어떤 규모의 조직이든 대부분 몇 가지 라이프사이클 단계를 거쳐 엔터프라이즈 IT 시스템을 배포하게 됩니다. 이 접근 방식을 때로 '스테이징 패턴'이라고도 합니다. 가장 흔한 패턴은 개발, 테스트, 프로덕션의 세 가지 단계입니다. 여러 단계를 거치면 애플리케이션과 애플리케이션 배포 변경 사항이 프로덕션 환경에 도달하기 전에 변경 사항을 안전한 환경에서 테스트할 수 있습니다. 가장 흔하고 권장되는 스테이징 패턴은 다음과 같이 서로 분리된 최소 3개의 Azure Red Hat OpenShift 클러스터를 마련하는 것입니다.

- **개발**: 모든 개발자와 운영자가 모든 것을 테스트할 수 있는 클러스터. 하나의 대규모 "샌드박스" 클러스터일 수 있지만 테스트가 자주 생성되고 소멸되는 작은 규모의 짧게 지속되는(short-lived) 클러스터를 마련하는 것이 더 일반적입니다. 또한 이렇게 하는 게 더 안전한 경우가 많습니다.
- **테스트**: 패치 또는 구성 변경과 같은 곧 있을 클러스터 변경 사항이 프로덕션으로 릴리스되기 전에 테스트 및 검증되는 클러스터. 일부 조직에서는 이를 가리켜 "사전 프로덕션"이라고 하는 경우도 많은데, 사전 프로덕션은 전체가 또 하나의 환경이 될 수도 있습니다.
- **프로덕션**: 실제 애플리케이션이 실행되는 클러스터.

이러한 환경 외에도 일부 조직에는 통합 테스트 환경과 같은 추가 환경을 보유하게 되지만 조직에 가장 적절한 스테이징 환경의 수는 고객님만 알 수 있습니다. 잘 모르겠다면 일반적인 배포 패턴에 맞는 다른 유사한 엔터프라이즈 애플리케이션을 찾아보세요.

Azure Red Hat OpenShift가 중요도가 낮은 애플리케이션에 사용되는 경우 2개의 클러스터(개발과 테스트 병합), 또는 개발, 테스트, 프로덕션을 포함하는 단일 클러스터만 조직 내에 보유하는 것도 가능합니다. 이렇게 하면 클라우드 비용을 낮추고 관리해야 하는 클러스터의 전체 개수를 줄일 수 있는 이점이 있습니다. 단일 클러스터에서 운영할 때 관리자는 개발, 테스트, 프로덕션 단계에 각기 별도의 네임스페이스를 사용하는 방식을 선택할 수 있습니다. 하지만 단일 클러스터를 실행하는 경우 다음과 같은 단점이 있습니다.

- 전체 클러스터(예: 소프트웨어 패치)에 영향을 미치는 변경 사항으로 인해 테스트 환경에서 실행했다면 찾아내서 방지할 수 있었을 문제가 프로덕션 단계에서 발생할 수 있습니다.
- 테스트 또는 개발 환경에서 애플리케이션이 다수의 컨테이너를 생성하거나 가용 디스크 공간을 모두 소진하는 등 예기치 않은 행동을 하는 경우 이러한 애플리케이션은 프로덕션 환경에서 문제를 야기합니다.

이 책에서는 고객님의 상황에 완벽히 들어맞는 클러스터 개수를 정확히 알려드릴 수 없지만, 앞서 언급했듯이 조직 내에서 유사한 엔터프라이즈 애플리케이션을 찾아보고 몇 개의 라이프사이클 단계를 사용하고 있는지 알아볼 수 있습니다.

비즈니스 연속성, 재해 복구, 페일오버

많은 조직은 표준 스테이징 환경과 더불어 최소 한 개의 페일오버 프로덕션 환경을 조성하는 방식에도 관심을 가질 것입니다. 페일오버 프로덕션 환경은 전체 클러스터 또는 Azure 지역을 중단시키는 심각한 장애가 발생했을 때 사용됩니다. 이를 가리켜 종종 **재해 복구(DR)**라고 합니다. 일반적으로 DR은 프로덕션 클러스터와는 다른 Azure 지역에 배포됩니다.

관리형 클라우드 서비스는 많은 비즈니스 크리티컬 애플리케이션에서 용인할 수 있는 99.95%의 **서비스 수준 계약(SLA)**과 함께 제공됩니다. 하지만 어떤 애플리케이션에는 더 높은 수준의 SLA가 필요할 수 있습니다. 단일 클러스터로는 이 SLA를 넘어설 수 없음을 이해하는 것이 중요합니다. 애플리케이션에 99.95%보다 높은 서비스 수준이 필요한지, 아니면 이 정도로 충분한지 생각해 보세요.

충분하지 않다면 여러 클러스터를 병렬로 실행하는 방식으로 복합 SLA를 계산하여 더 높은 수준의 서비스 가용성(예: 99.999%)에 도달할 수 있습니다. 클러스터는 모두 동일한 지역 내에 있거나(예: `westeurope`) 훨씬 더 높은 가용성 수준을 위해 여러 지역에 있을 수 있습니다(예: `westeurope` 및 `northeurope`). 다음 Azure 도큐멘테이션에서는 여러 개의 클러스터를 실행할 때 복합 SLA를 계산하는 방법에 대해 설명합니다.

복합 SLA에 관한 Azure 도큐멘테이션

Azure Red Hat OpenShift의 다중 클러스터 및 다중 지역 배포는 이 책의 범위를 벗어난 주제입니다. 왜냐하면 이처럼 더 복잡한 아키텍처를 구축할 때 트래픽을 클러스터로 이동하는 것과 관련해 몇 가지 해결할 과제가 있고 클러스터 간 데이터 공유 방법과 공유할 데이터를 선택할 때 고려할 사항이 많기 때문입니다.

지역과 가용성 영역

Azure Red Hat OpenShift는 배포되는 각 지역에서 세 개의 가용성 영역을 활용하도록 설계되었습니다. Azure에서 가용성 영역이란 자체 전력, 냉각, 네트워크 연결성을 보유한 지역 내 자율 데이터센터를 말합니다. Azure Red Hat OpenShift 배포를 검사하는 경우 각 가용성 영역에서 단일 제어 노드(가상 머신) 및 애플리케이션 노드가 보일 것입니다.

*그림 4.1*은 한 곳의 Azure 지역 내에서 제어 노드와 애플리케이션(작업자) 노드가 세 곳의 가용성 영역에 걸쳐 어떻게 분산되어 있는지 보여줍니다.

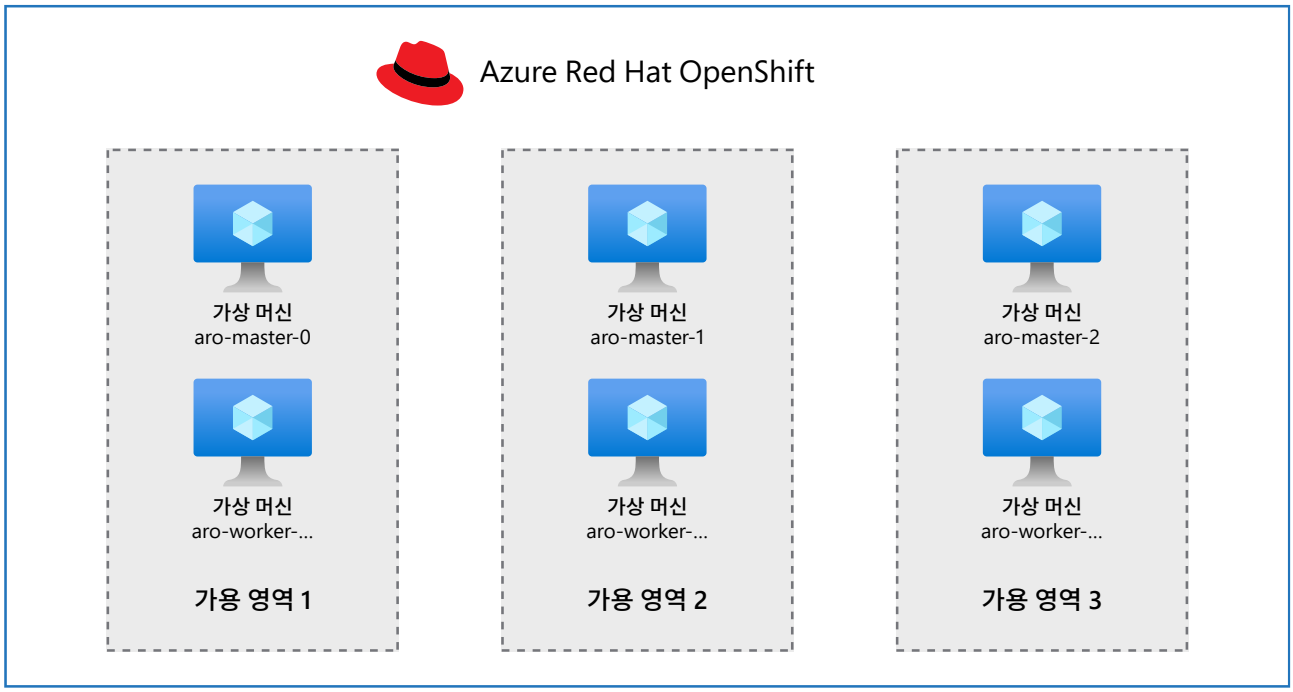


그림 4.1: 한 곳의 Azure 지역 내 세 곳의 가용성 영역에 걸쳐 분산되어 있는 제어 및 애플리케이션 노드

Azure Red Hat OpenShift는 완전 관리형 고가용성 서비스로 제공되도록 설계되었습니다. 따라서 제어 노드와 작업자 노드를 각기 세 개 미만으로 배포하는 것은 불가능합니다.

네트워크 개념

다음 Microsoft 도큐멘테이션 사이트에서도 볼 수 있는 네트워킹 개념에 관한 이 탁월한 도큐멘테이션은 앞서 Azure Red Hat OpenShift를 소개하는 섹션에서도 언급해 드렸습니다.

- Azure Red Hat OpenShift를 위한 네트워크 개념

이 페이지에는 로드 밸런서, 퍼블릭 및 프라이빗 IP 주소, 네트워크 보안 그룹 등 Azure Red Hat OpenShift의 모든 네트워킹 구성 요소에 대한 자세한 설명이 포함되어 있습니다.

알아두어야 할 몇 가지 핵심 포인트는 다음과 같습니다.

- Azure Red Hat OpenShift는 기존 또는 신규 가상 네트워크로 배포됩니다. 하나의 가상 네트워크도 지원되지만 OpenShift가 OVS라고 하는 자체 **소프트웨어 정의 네트워크(SDN)**를 최상위에 배치하기 때문에 네트워크가 여러 개라 하더라도 부가적인 이점은 없습니다.
- 마스터 및 애플리케이션 노드 서브넷의 최소 크기는 /27입니다.
- 기본 포드 CIDR은 10.128.0.0/14입니다.
- 기본 서비스 CIDR은 172.30.0.0/16입니다.
- 각 노드는 포드에 대해 /23 서브넷(512개의 IP 주소)이 할당되어 있습니다. 이 값은 변경할 수 없습니다.
- 이그레스 IP는 현재 지원되지 않습니다.
- 이그레스 트래픽 라우팅을 특히 Azure Firewall을 통해 전송할 수 있도록 제어할 수 있습니다. 이 문서를 작성하는 시점에 이 기능은 공개 미리 보기로 제공되고 있으며 이그레스 트래픽 제어라는 문서로 정리되어 있습니다.

퍼블릭 또는 프라이빗 네트워크 가시성

Azure Red Hat OpenShift를 퍼블릭 또는 프라이빗 배포로 배포할 수 있다는 얘기를 종종 들을 수 있습니다. 맞는 말이긴 하지만 컨트롤 플레인을 퍼블릭/프라이빗으로 만들고 클러스터의 애플리케이션을 퍼블릭/프라이빗으로 만드는 것의 차이를 이해하면 도움이 됩니다.

API 서버 가시성의 경우 프로비저닝 시점에 해야 할 의사 결정입니다. 클러스터가 프로비저닝된 후에는 퍼블릭/프라이빗 가시성을 조정하는 것이 불가능합니다.

나중에 이 장에서 Azure Red Hat OpenShift 클러스터를 생성하게 되면 클러스터의 가시성에 관한 인수를 받아들이는 `az aro create` 커맨드를 실행하게 됩니다. 아래의 예시는 가시성을 조정하는 방법을 보여줍니다.

둘 다 프라이빗

```
az aro create .... --apiserver-visibility Private --ingress-visibility Private
```

프라이빗 API 서버와 퍼블릭 작업자 인그레스

```
az aro create .... --apiserver-visibility Private --ingress-visibility Public
```

apiserver 및 애플리케이션 인그레스에 대한 가시성은 *그림 4.2*의 Azure 아키텍처 다이어그램으로 매핑됩니다. 이 다이어그램에서는 Azure Red Hat OpenShift가 내부 및 퍼블릭 Azure 로드 밸런서를 사용하는 방식을 보여줍니다. 이 경우 API와 라우터는 선택한 가시성 옵션에 따라 배포됩니다.

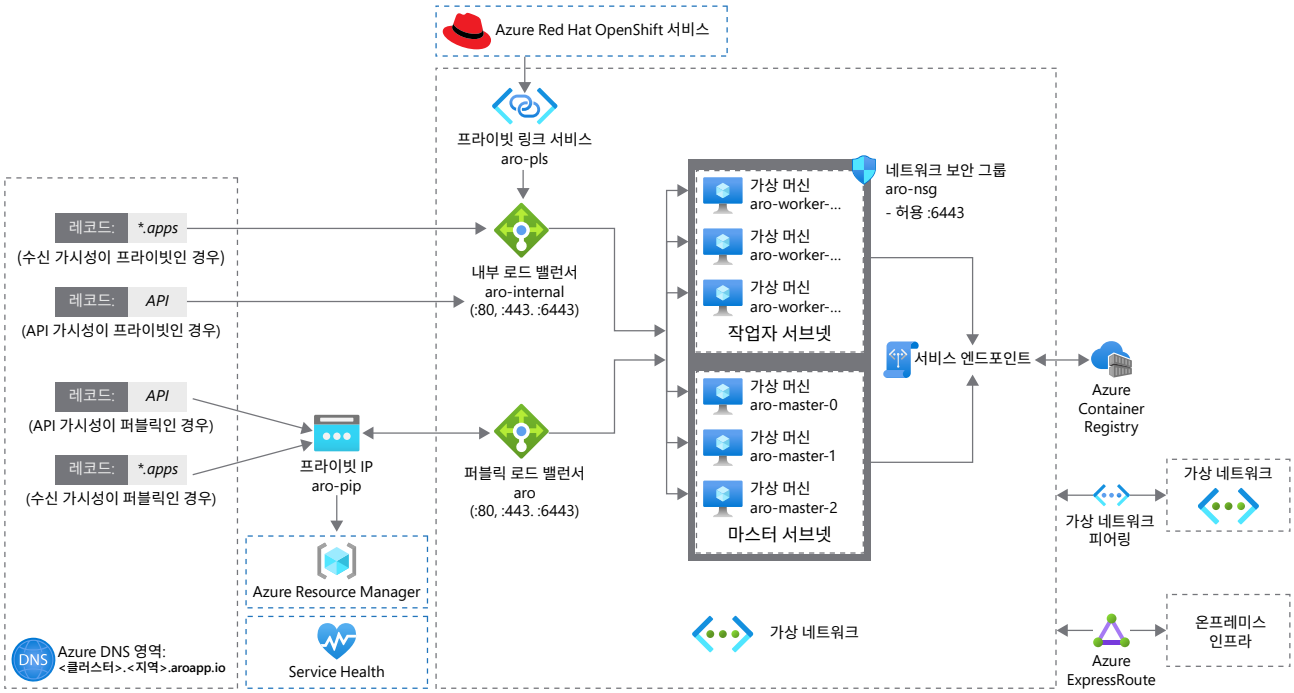


그림 4.2: 내부 및 퍼블릭 Azure 로드 밸런서를 사용하는 Azure Red Hat OpenShift

API 서버 가시성 및 인그레스 가시성에 관한 더 자세한 설명은 다음과 같습니다.

API 서버 가시성(컨트롤 플레인)

`--apiserver-visibility`는 **퍼블릭** 또는 **프라이빗**일 수 있습니다.

- **프라이빗**은 쿠버네티스 API가 실행되는 Red Hat OpenShift 컨트롤 플레인(이전 명칭은 "마스터 노드")에 퍼블릭 인터넷을 통해 액세스할 수 없음을 뜻합니다. 이 컨트롤 플레인은 개발자와 운영자가 클러스터를 제어하는 것이 목적이며, 클러스터 자체뿐 아니라 애플리케이션까지도 배포, 삭제 또는 확장하는 데 사용할 수 있습니다. Azure에 대한 명시적 경로 연결을 보유하고 있거나 **가상 프라이빗 네트워크(VPN)**를 사용해 컴퓨팅 리소스에 액세스하는 기업은 대부분의 경우 프라이빗을 선택해야 합니다.
- **퍼블릭**은 클러스터 컨트롤 플레인에 퍼블릭 인터넷을 통해 액세스할 수 있음을 뜻합니다. 따라서 클러스터는 액세스에 대한 인증이 이루어지더라도 인터넷상에 있는 누군가로부터 공격을 받을 위험에 노출됩니다. 하지만 이를 퍼블릭으로 설정하면 사용자의 소스 네트워크를 제어할 수 없는 랩 또는 테스트 환경에 유용합니다. 실제 데이터가 저장되는 환경이나 모든 종류의 프로덕션 환경에서는 API 서버 가시성을 프라이빗으로 설정할 것을 적극 권장합니다.

아래 나열된 항목은 API 서버 연결이 필요한 몇 가지 예시입니다. 퍼블릭 또는 프라이빗 중에 선택할 때는 다음 사항을 고려해야 합니다.

- IDE에서 스크립트 및 툴을 사용하는 개발자(예: kubectl rollout)
- 클러스터의 상태를 검사하는 운영자(예: kubectl get nodes)
- 배포의 상태를 검사 또는 조정해야 하는 CI/CD 서버(예: Azure DevOps)
- 상태를 검토하기 위해 클러스터에 연결하는 클러스터 보안 툴
- 쿠버네티스 API에 의존하는 모니터링 툴

대부분의 경우 네트워킹은 **프라이빗** 연결을 통해 제공되도록 구성할 수 있지만 위에 나열된 항목에 조직 내부의 추가 예시가 포함될 수 있습니다. **퍼블릭** API 서버 가시성에 대한 예기치 않은 요구 사항이 발생할 수 있습니다.

인그레스 가시성(애플리케이션)

`--ingress-visibility`는 퍼블릭 또는 프라이빗일 수 있습니다.

- **프라이빗**은 노출된 서비스(클러스터에서 실행되는 애플리케이션과 관련이 있음)가 퍼블릭 인터넷에서 직접 연결할 수 있도록 허용하지 않음을 뜻합니다. 애플리케이션에 연결하기 전에 별도의 Azure 네트워크에서 선택적으로 실행되는 Azure Firewall 또는 웹 애플리케이션 방화벽을 사용자가 먼저 통과할 수 있도록 네트워킹 라우팅을 구성하는 것이 여전히 가능합니다. **프라이빗**은 클러스터의 애플리케이션이 조직 내부 용도로 제작된 경우에도 잘 작동합니다(예: 급여 처리, 데이터 분석 또는 기타 내부 웹 애플리케이션).
- **퍼블릭**은 클러스터의 애플리케이션에 퍼블릭 인터넷을 통해 도달할 수 있음을 뜻합니다. 하지만 이러한 애플리케이션에 액세스하려면 여전히 인그레스 또는 경로 리소스를 구성해야 합니다. 이는 Red Hat OpenShift에서 퍼블릭 전자 상거래 쇼핑 웹사이트나 기타 퍼블릭 애플리케이션을 호스팅하는 경우에도 마찬가지입니다.

인그레스를 가리켜 때로 OpenShift "라우터"라고 합니다.

프로덕션 관련 권장 사항

다음 사항을 적극 권장합니다.

- 퍼블릭 인터넷이 아닌 프라이빗 연결을 통해 클러스터에 액세스하세요. Azure ExpressRoute는 온프레미스 네트워크 또는 기업 사무실로부터의 영구적 연결이 클러스터에 필요할 때 가장 좋은 선택지입니다. 또는 Azure에 대한 VPN으로도 프라이빗 연결을 제공할 수 있습니다. 이에 관한 자세한 내용은 **하이브리드 연결성** 섹션에서 볼 수 있습니다.
- API 서버 가시성(컨트롤 플레인)을 프라이빗으로 설정하고, 선택적으로 방화벽을 이용해 액세스 권한을 더 제한하세요.
- 조직 내에서 실행되는 애플리케이션의 인그레스 가시성(애플리케이션)을 프라이빗으로 설정하세요. 퍼블릭 인터넷에서 호스팅되어야 하는 Azure Red Hat OpenShift에 기반을 둔 애플리케이션의 활용 사례가 있는 경우 별도의 Azure Red Hat OpenShift 클러스터를 설정하세요. 내부 애플리케이션에 최소 하나, 그리고 인그레스 가시성이 퍼블릭으로 설정된 외부 애플리케이션에 하나를 설정합니다. 하지만 동일한 클러스터 내에서 퍼블릭 및 프라이빗 인그레스를 혼합할 수 있습니다.

물론 많은 조직이 자체 Azure 네트워킹 및 보안 팀과 상의해 필요할 수 있는 추가 컨트롤을 결정하게 됩니다. 예를 들어 어떤 조직은 웹 애플리케이션 앞에 웹 애플리케이션 방화벽을 배치해야 합니다. 이는 Azure Red Hat OpenShift에 애플리케이션을 배포할 때도 일반적인 배포 패턴입니다.

하이브리드 연결성

Azure Red Hat OpenShift를 배포하는 대부분의 조직은 온프레미스 환경에서 실행되는 지원 서비스로 다시 연결하는 데 필요한 애플리케이션을 실행할 것입니다. Azure에서 온프레미스로 다시 연결하는 경우 이를 가리켜 일반적으로 하이브리드 연결성 또는 하이브리드 클라우드 아키텍처라고 합니다. 온프레미스 환경으로 다시 연결성을 제공하는 방법이 몇 가지 있습니다. 가장 중요한 방법은 다음 두 가지입니다.

- **VPN**: Azure에 대한 복잡성이 낮은 연결에 적합합니다. 일반적으로 VPN은 Azure VPN Gateway를 통해 연결됩니다. 자세한 내용은 Azure VPN Gateway란?을 참조하세요.
- **Azure ExpressRoute 회로**: Azure에 대한 강력하고 영구적인 전용 연결에 적합합니다. 자세한 내용은 Azure ExpressRoute란?을 참조하세요.

어떤 연결 방법을 사용하든 두 솔루션 모두 온프레미스의 애플리케이션이 Azure Red Hat OpenShift에서 실행되는 애플리케이션에 연결하거나 그 반대 방향으로 연결하는 것을 허용할 수 있습니다.

온프레미스 환경에서 Azure Red Hat OpenShift에 연결할 때 허브 가상 네트워크를 통해 연결하는 것이 일반적입니다 Azure ExpressRoute를 통해 연결되는 방식을 설명하는 다이어그램이 *그림 4.3*에 표시되어 있습니다.

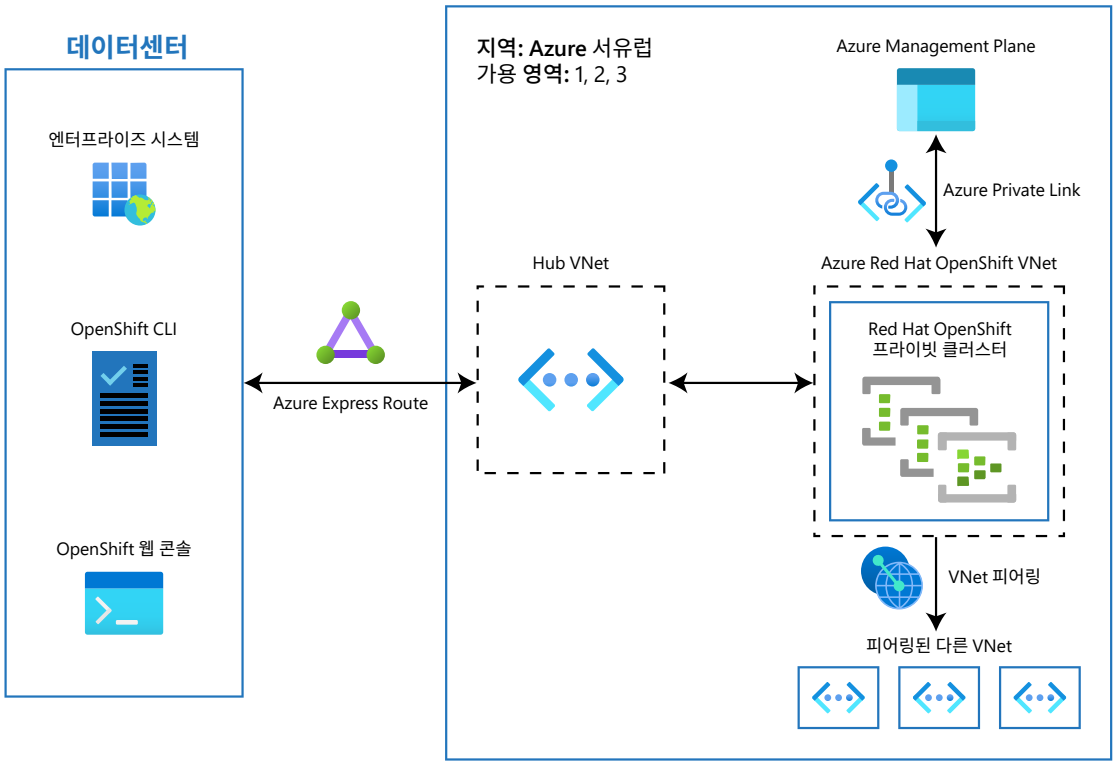


그림 4.3: Azure ExpressRoute를 통한 연결

위 다이어그램에서는 Azure Red Hat OpenShift에 연결되는 Azure ExpressRoute의 개략적인 아키텍처를 보여줍니다. 이 다이어그램에서는 Azure ExpressRoute를 보여주지만 VPN을 통한 연결은 개념적으로 이와 매우 유사합니다.

하이브리드 아키텍처에서 실행되는 애플리케이션과 관련해 고려할 사항

온프레미스 환경에 다시 연결되는 Azure Red Hat OpenShift에서 애플리케이션을 실행하는 경우 애플리케이션 소유자가 맞닥뜨릴 수 있는 몇 가지 고려 사항이 있습니다.

- **연결 대기 시간:** Azure ExpressRoute는 온프레미스 환경에 대해 대기 시간이 비교적 짧은 연결을 제공하는 반면, 퍼블릭 인터넷을 통해 트래픽을 전송하는 VPN 네트워크는 대기 시간이 상당히 더 길 수 있습니다. 웹 서버와 같이 연결이 HTTP 트래픽만 전송하는 일부 애플리케이션의 경우 문제를 일으킬 가능성이 높지 않습니다. 하지만 이 웹 서버에서 실행되는 서버 측 애플리케이션을 온프레미스에서 실행되는 데이터베이스에 연결해야 하는 경우 성능에 상당한 영향을 미칠 수 있습니다.
- **인그레스/이그레스 트래픽에 드는 비용:** 일부 연결 유형, 즉 Azure 또는 연결 제공업체를 통한 인그레스 및 이그레스 트래픽에는 요금이 청구됩니다. 합리적인 예방책은 먼저 테스트 환경에서 비용을 측정한 다음, 피크 로드에서는 비용이 얼마일지 예상해 봄으로써 당황스러운 일을 겪지 않도록 하는 것입니다.
- **장애 시나리오:** Azure ExpressRoute 연결은 영구적이고 강력한 성능을 발휘하도록 설계된 반면, VPN 연결은 중단되거나 연결/연결 해제되는 일을 자주 겪습니다. 또한 퍼블릭 인터넷을 통해 실행되기 때문에 VPN 연결 대기 시간과 서비스 품질은 하루 중에 꽤 자주 변경될 수 있습니다. 하이브리드 연결 조건이 열악할 때뿐 아니라 연결이 최적의 성능으로 실행될 때도 애플리케이션 성능을 테스트하는 것이 중요합니다. 또한 몇 시간 동안 연결이 중단되는 경우 Azure Red Hat OpenShift에서 실행되는 애플리케이션은 저하된 성능으로 계속 실행될까요, 아니면 애플리케이션의 가용성은 전적으로 하이브리드 연결에 따라 달라질까요?

조직의 기존 내부 체크리스트에 추가해야 할 몇 가지 다른 항목이 있을 수 있지만 앞서 언급한 항목만으로도 하이브리드 아키텍처가 어떻게 작동할지 고려하기에 충분합니다.

요약

이 장에서는 Azure Red Hat OpenShift 배포 전에 묻고 따져봐야 할 일반적인 여러 가지 프로비저닝 전 질문에 대해 알아보았습니다. 다음 장에서는 클러스터를 실제 배포할 때 참고할 수 있는 리소스에 대한 유용한 조언을 제공해 드립니다.

5장

Azure Red Hat OpenShift 클러스터 프로비저닝

지금까지 알아본 내용을 다시 간략히 살펴보겠습니다. 이 책에서 다룬 내용은 다음과 같습니다.

- *2장: Red Hat OpenShift 소개*에서는 Red Hat OpenShift에 대해 간략히 소개하고 기본 쿠버네티스 대신 OpenShift를 선택할 경우 얻을 수 있는 이점에 대해 설명했습니다.
- *3장: Azure Red Hat OpenShift*에서는 Azure Red Hat OpenShift 관리형 클라우드 서비스에 관해 세부적으로 설명했고 이 서비스의 아키텍처, 관리, 인증, 지원, 가격 책정 관련 고려 사항을 포함한 핵심 개념을 다루었습니다.
- *4장: 프로비저닝 전 – 엔터프라이즈 아키텍처에 관한 질문*에서는 조직이 Azure Red Hat OpenShift를 배포하기 전에 해결할 일반적인 질문에 대해 알아보았습니다.

이제 클러스터를 프로비저닝하고 배포할 준비가 되었으므로 도큐멘테이션 사이트에서 공식적인 배포 지침을 확인하시기 바랍니다(프로비저닝은 배포 과정의 일부이며, 배포를 프로비저닝한다는 것은 배포를 지원하는 데 필요한 IT 인프라가 마련되도록 보장하는 것을 뜻함).

튜토리얼 – Azure Red Hat OpenShift 4 클러스터 생성

이 장에서는 클러스터를 생성하기 위해 입력해야 하는 개별 커맨드는 다루지 않습니다. 이러한 커맨드는 시간이 지나면서 변경되게 마련이고 도큐멘테이션에서 이미 자세히 다루고 있기 때문입니다.

하지만 이 장에서는 전체 프로세스에 관한 지침과 클러스터 배포 시 고려할 사항을 제공합니다.

수동 배포 – 시간 기대치

어떤 조직의 경우 클러스터를 프로비저닝하는 데 걸리는 시간을 파악해야 합니다. 이에 대해 자세히 설명드리겠습니다.

배포를 위한 사전 요건의 개략적인 프로세스는 다음과 같습니다.

- 시스템 관리자의 워크스테이션에 배포되는 az 커맨드라인
- Azure 서브스크립션과 함께 사용할 수 있도록 등록된 리소스 제공자
- Red Hat 풀(pull) 암호(선택 사항)
- 클러스터용 도메인(선택 사항)
- 컨트롤 플레인 노드와 애플리케이션 노드에 빈 서브넷이 각기 한 개씩 있는 가상 네트워크

이러한 사전 요건을 설정하는 데 필요한 시간의 측면에서 숙련된 Azure 및 Red Hat OpenShift 관리자는 아마도 처음에는 이러한 태스크를 30분 내로 완료할 수 있을 것입니다. 이러한 단계를 수동 프로세스의 일부로 반복 실행한다면 10분 내로 완료할 수 있을 것입니다.

사전 요건이 충족되었다면 자동 프로비저닝 프로세스는 Azure 지역 내 활동에 따라 일반적으로 25~40분 소요됩니다.

배포 자동화

Azure Red Hat OpenShift 프로비저닝 프로세스가 자동화된다는 것을 이해한 조직은 툴을 사용해 네트워크, 서브넷, 서비스 원칙 등을 생성하는 사전 요건 단계도 자동화하려고 하는 것이 일반적입니다.

이러한 단계들을 자동화할 수 있는 여러 툴이 있습니다. 몇 가지 권장 툴은 다음과 같습니다.

- az 커맨드라인 툴: 자동화된 경우 이 툴은 일반적으로 CI/CD 프로세스의 일부로 컨테이너 또는 이와 유사한 것에 설치됩니다. 여기에서 사용되는 일반적인 툴로 Jenkins, Azure DevOps 또는 Ansible을 들 수 있습니다. az 커맨드라인 툴은 한 번만 배포하면 되지만 추가 클러스터는 조직의 여러 부분을 반영하도록 Azure 서브스크립션 ID를 설정해야 할 수 있습니다.
- Azure 서브스크립션과 함께 사용할 수 있도록 등록된 리소스 공급자: 앞서 언급한 것처럼 az 커맨드라인 툴 설정의 일부입니다.
- Red Hat 풀(pull) 암호(선택 사항): Red Hat은 풀(pull) 암호 획득을 위해 지원되는 REST API를 문서화했습니다. 이에 관한 정보는 이 문서에서 보실 수 있습니다.
- 클러스터용 도메인(선택 사항): DNS 레코드를 생성하는 방식에 따라 달라집니다. 하지만 Azure DNS를 사용하는 경우 Terraform, Ansible을 비롯하여 널리 사용되는 다른 Azure 자동화 툴이 자동으로 수행할 수 있습니다.
- 컨트롤 플레인(마스터) 노드와 애플리케이션(작업자) 노드에 빈 서브넷이 각각 한 개씩 있는 가상 네트워크: 일반적으로 널리 사용되는 Azure 자동화 툴에 의해 자동화됩니다. 즉, Terraform, Ansible 또는 이와 유사한 툴은 이 네트워크와 서브넷을 사용자를 자동으로 생성할 수 있습니다.

사전 요건 단계가 자동화되면 이 시간을 30분 내지 10분으로 단축할 수 있습니다. 수동 프로세스인 경우에는 단 1~2분으로 단축하는 것도 가능합니다. 클러스터 배포 속도를 높이는 것은 불가능하지만(항상 25~40분 걸림) 온프레미스에 비하면 매우 빠른 엔드 투 엔드 배포 프로세스일 수 있습니다.

배포 자동화는 프로세스 속도를 높이는 것 이상의 여러 이점이 있습니다. 배포 자동화를 사용하는 고객은 손쉽게 로깅하고 감사할 수 있는 강력하고 반복 가능한 프로세스를 구축할 수 있는 이점을 누리게 됩니다. 고객이 셀프 서비스 카탈로그 항목을 자체 포털에 구성하는 것은 지극히 흔한 일입니다. 이를 통해 팀은 클라우드 플랫폼 팀에서 상호 작용 없이도 Azure Red Hat OpenShift 클러스터를 손쉽게 프로비저닝 및 프로비저닝 해제할 수 있습니다.

클러스터에 액세스

이 섹션에서는 Azure Red Hat OpenShift 클러스터를 프로비저닝한 후 액세스하는 방법에 관해 간단한 참조할 수 있는 내용을 제공합니다.

웹 UI를 통한 액세스

CLI를 설치한 경우 Bash 셸에서, CLI를 설치하지 않은 경우 Azure Portal의 Azure Cloud Shell(Bash) 세션에서 다음 커맨드를 실행하여 클러스터 로그인 URL을 검색합니다.

```
az aro show -n $CLUSTER_NAME -g $RG_NAME --query “consoleProfile.url” -o tsv
```

`openshift.xxxxxxxxxxxxxxxxxxxx.eastus.aroapp.io`와 같은 결과가 반환되어야 합니다. 클러스터의 로그인 URL은 `https://` 다음에 `consoleProfile.url` 값이 옵니다 (예: `https://openshift.xxxxxxxxxxxxxxxxxxxx.eastus.aroapp.io`).

이 URL을 브라우저에서 엽니다. `kubeadmin` 사용자로 로그인하라는 메시지가 표시됩니다. 설치 프로세스 중에 설치 프로그램이 제공한 사용자 이름과 비밀번호를 사용합니다.

로그인하면 Azure Red Hat OpenShift 웹 콘솔이 표시되어야 합니다.

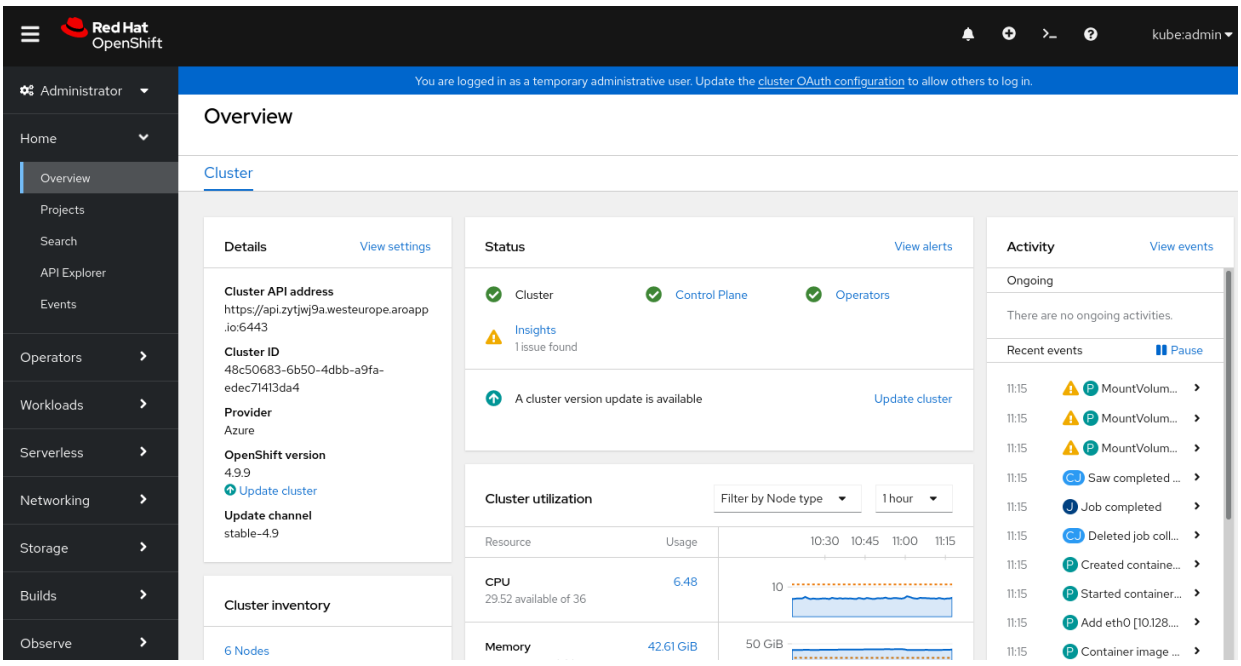


그림 5.1: Azure Red Hat OpenShift 웹 콘솔

여유를 갖고 웹 콘솔을 두루 살펴봅니다. 웹 콘솔에서 OpenShift 최신 버전이 실행 중이어야 하고 모든 구성 요소가 양호한 상태에 있어야 합니다. 또는 설치 후 곧 양호한 상태로 안정화됩니다.

OpenShift CLI(oc)를 통한 액세스

최신 OpenShift CLI(oc)를 다운로드해야 합니다. 이를 위해서는 로그인하여 https://console.redhat.com/openshift/downloads 페이지를 확인하면 됩니다.

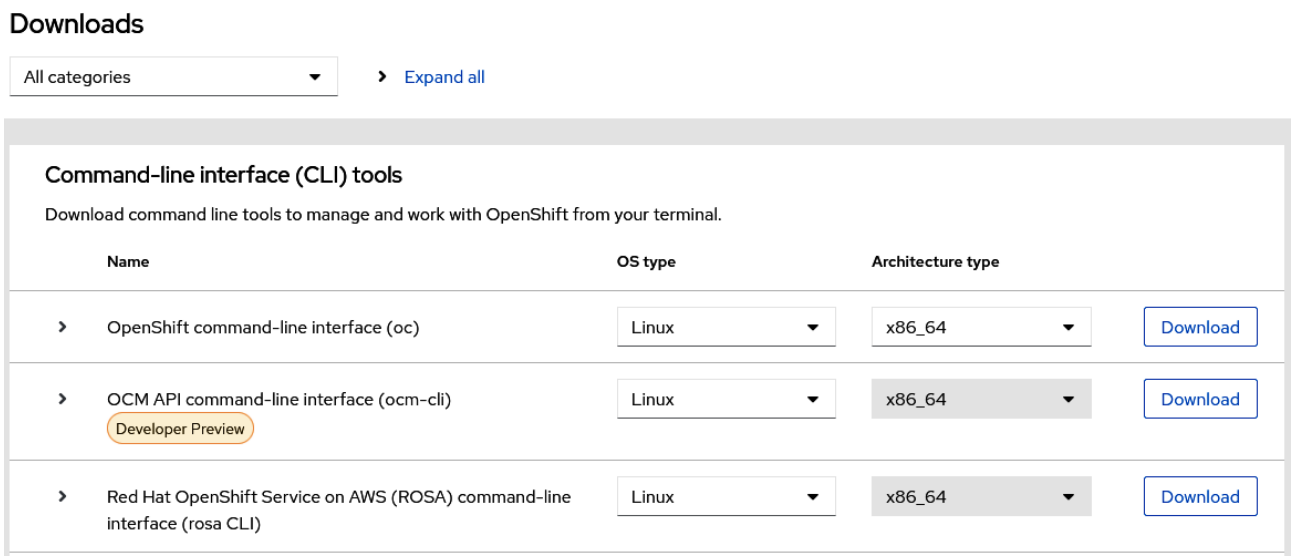


그림 5.2: OpenShift 커맨드라인 인터페이스 다운로드

아카이브(`.tar.gz` 또는 `.zip`)를 시스템의 특정 위치에 푼 다음, oc 커맨드를 경로의 특정 위치에 배치합니다. Linux에서는 oc 커맨드를 `/usr/local/sbin/`에 배치하는 것이 꽤 일반적입니다.

OpenShift CLI 실행 및 클러스터 로그인

커맨드라인에서 클러스터에 인증하려면 웹 콘솔에서 로그인 커맨드 및 토큰을 검색해야 합니다. 브라우저에서 웹 콘솔에 로그인하여 오른쪽 상단의 사용자 이름을 클릭한 후 **로그인 커맨드 복사**를 클릭합니다.

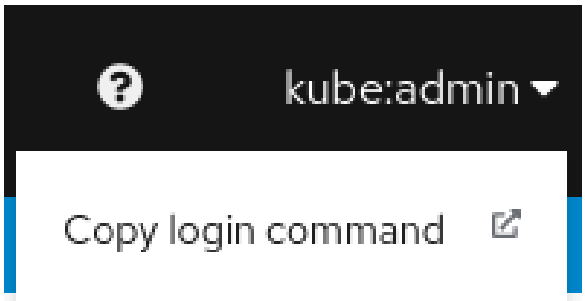


그림 5.3: 로그인 커맨드 복사

그러면 다음과 같은 페이지가 새로 열립니다.

그림 5.4: API 토큰으로 로그인하기

그런 다음 이 커맨드를 복사해 터미널에 붙여넣어 Azure Red Hat OpenShift 클러스터에 로그인할 수 있습니다.

예를 들어 Azure Portal에서 Azure Cloud Shell(Bash) 세션을 사용 중인 경우 로그인 커맨드를 붙여넣으면 `oc status` 커맨드가 다음과 같이 표시됩니다.

```
user@Azure: oc status
In project default on server https://api.cyki1k6g.westeurope.aroapp.io:6443

svc/openshift - kubernetes.default.svc.cluster.local
svc/kubernetes - 172.30.0.1:443 -> 6443

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.
```

이제 여유를 갖고 oc 커맨드라인을 더 살펴보면 새로운 Azure Red Hat OpenShift 환경에 친숙해질 수 있습니다. 이미 OpenShift 4를 자주 사용하셨다면 Azure Red Hat OpenShift의 이 클라우드 서비스가 매우 친숙할 것이므로 과거에 사용했을 수 있는 다른 OpenShift 4 환경과 동일한 방식으로 작동할 것입니다.

요약

매우 간략한 이 장은 프로비저닝에 관한 공식 지침으로 잘 유지 관리되고 있으므로 Azure Red Hat OpenShift를 Azure 서브스크립션에서 실행할 때 확정적인 가이드로 참고해야 합니다. 프로비저닝 지침은 자체 관리형 OpenShift 환경을 프로비저닝하는 데 필요한 지침보다 훨씬 단순하다는 것을 알 수 있을 겁니다. 이는 상당한 양의 엔지니어링 작업이 Azure Red Hat OpenShift 서비스에 투입되어 Azure와의 긴밀한 통합을 제공하고 사전 규정된 "모든 상황에 적합한" 아키텍처(테스트 및 지원이 제대로 이루어짐)를 배포하기 때문입니다. 전반적으로 이는 조직에 이익이 됩니다. Azure Red Hat OpenShift 프로비저닝에 소요되는 시간이 단축되므로 애플리케이션 배포에 더 많은 시간을 사용할 수 있기 때문입니다.

6장

프로비저닝 후 – 2일 차

Azure Red Hat OpenShift를 배포하고 나면 클러스터를 프로덕션 단계에 맞게 준비하기 전에 일반적으로 필요한 몇 가지 프로비저닝 후 활동을 수행해야 합니다. 이 장에서는 다음과 같은 일반적인 프로비저닝 후 여러 가지 활동을 다룹니다.

- 인증 – Azure Active Directory – 사용자 그룹을 커뮤니티 운영자와 동기화하는 방법 포함
- 운영자 – OperatorHub 포함
- 로깅에 대한 이해 – 로그 전달 포함
- 모니터링에 대한 이해 – OpenShift 컨테이너 모니터링 포함
- 업그레이드 및 패치 – 지원되는 버전 라이프사이클 포함
- 클러스터 확장 – 클러스터 수동 및 자동 확장 포함
- 애플리케이션 확장 – 애플리케이션 확장에 관한 간략한 참고 사항
- 풀(pull) 암호 설정 – OpenShift Cluster Manager에 등록
- 제한 범위 – 제한 범위를 적용할 수 있는 경우 포함
- 퍼시스턴트 스토리지 – 사용 가능하고 지원되는 스토리지 클래스 포함
- 보안 및 컴플라이언스 – 보안 제어에 관한 참고 사항

다양한 프로비저닝 후 태스크에 관해 자세히 다루는 이 각 섹션을 통해 새로 배포된 클러스터를 받아들여 프로덕션 애플리케이션에 맞게 준비시키기 위해 필요한 것이 무엇인지 더 깊이 이해할 수 있습니다.

인증 – Azure Active Directory

Azure Red Hat OpenShift는 OpenShift 도큐멘테이션에 나열된 모든 인증 공급자를 지원합니다. 인증 공급자 전체 목록을 확인하세요. 하지만 대다수 고객은 이미 Azure에서 제공하는 Azure Active Directory를 사용해 Azure Red Hat OpenShift에 인증과 SSO(Single Sign-On)를 제공할 가능성이 높습니다.

Azure Active Directory 통합을 위한 구성은 조직이 다른 구성 제공업체를 사용하길 원하는 경우가 있을 수 있으므로 의도적으로 "2일 차" 운영입니다. Azure Active Directory 설정 및 설치 프로세스는 처음 설정할 때 약 15분 소요되며, 이 프로세스에 더 익숙해지면 5분 내로 완료할 수 있습니다. 많은 조직이 ARM 템플릿 또는 Ansible Playbook과 같은 툴을 사용해 이 프로비저닝 과정의 일부를 자동화하고자 합니다.

- Azure Red Hat OpenShift에 맞게 Azure Active Directory 구성(그래픽 포털)
- Azure Red Hat OpenShift에 맞게 Azure Active Directory 구성(커맨드라인 인터페이스)

Active Directory 사용자 그룹 사용

Azure Active Directory 인증을 구성하면 사용자가 기존 자격 증명으로 로그인하는 것만 가능합니다. 사용자의 기존 사용자 그룹을 Red Hat OpenShift로 가져오지는 못합니다. 조직이 사용자 그룹을 사용해 OpenShift 내에서 사용자 권한을 구성하기 위해 Active Directory에서 사용자 그룹을 가져오길 원하는 경우가 꽤 흔합니다. 각각의 커뮤니티 지원 오퍼레이터는 그룹 동기화 오퍼레이터 역할을 할 수 있습니다.

- GitHub의 그룹 동기화 오퍼레이터

그룹 동기화 오퍼레이터는 커뮤니티의 지원을 받습니다. 이는 이 문서 작성 시점에는 Red Hat이나 Microsoft의 공식 지원을 받지 않음을 뜻합니다. 이 책에서는 프로젝트의 README 파일과 함께 제공되는 오퍼레이터용 세부 구성 및 설정 지침을 따를 것을 권장합니다.

오퍼레이터

OpenShift 4의 오퍼레이터는 플랫폼을 구성하는 가장 기본적인 요소 중 하나로서, Red Hat OpenShift 소비자에게 막대한 가치를 제공합니다. 오퍼레이터는 코드로 빌드되며 클러스터의 컨테이너에서 서비스를 실행합니다. 어떤 오퍼레이터는 클러스터 네트워킹, 머신 구성 유지 관리, 클러스터 업그레이드와 같은 작업을 유지 관리할 책임이 있습니다. 이를 가리켜 클러스터 오퍼레이터라고 하는 경우가 많은데, OpenShift 콘솔의 **관리** 섹션에서 이러한 오퍼레이터의 목록을 볼 수 있습니다.

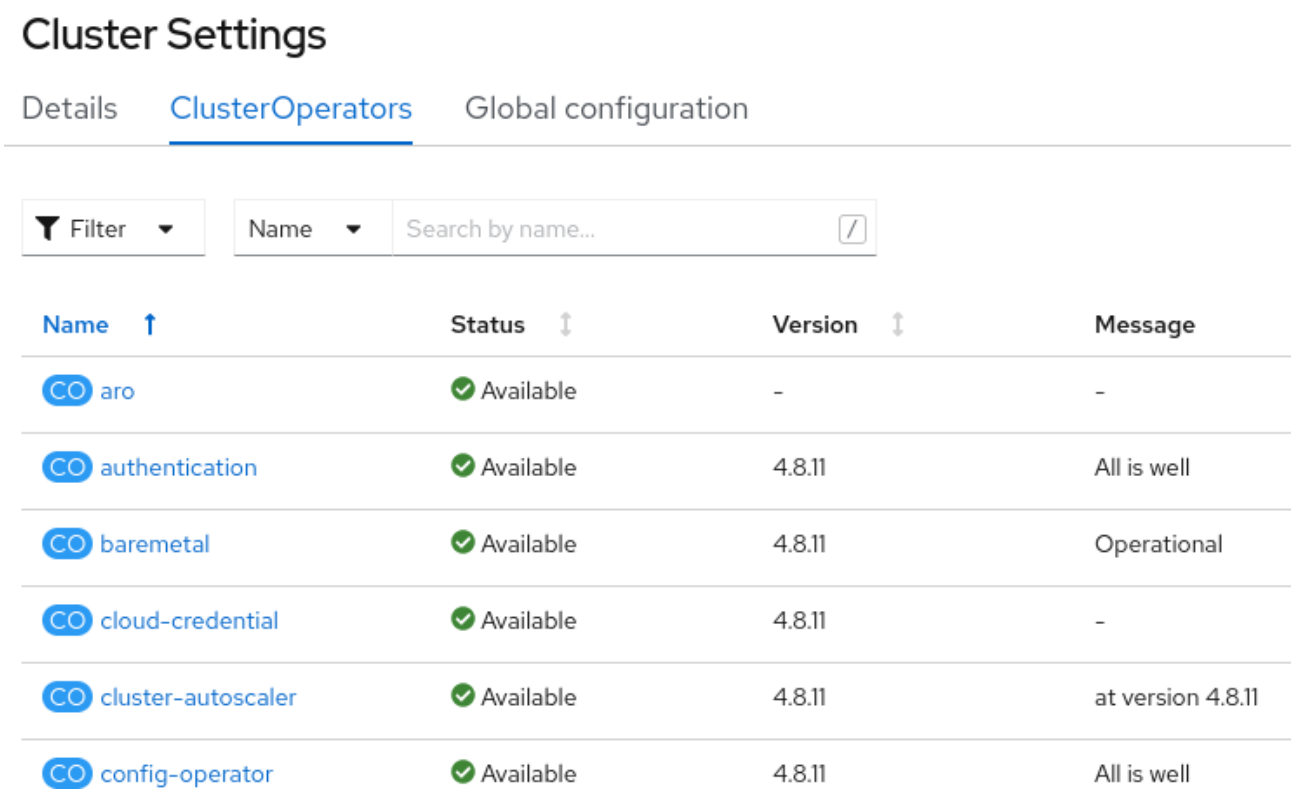


그림 6.1: 클러스터 설정

위 스크린샷에서 Azure Red Hat OpenShift만을 위한 오퍼레이터가 있음을 알 수 있는데, 이 오퍼레이터는 서비스의 여러 부분을 유지 관리하고 지원되는 구성 상태에 클러스터가 머물도록 보장하기 위해 노력합니다.

오퍼레이터는 서비스 상태, 업데이트, 패치, 확장, 기타 몇 가지 기능과 같은 여러 작업을 유지 관리할 수 있습니다.

OperatorHub

관리자는 모든 클러스터의 백그라운드에서 실행되는 표준 클러스터 오퍼레이터 외에도 OperatorHub를 통해 더 많은 오퍼레이터에 액세스할 수 있습니다. OperatorHub를 사용하면 관리자가 Red Hat OpenShift 설치와 관련해 제공하고 싶은 인기 있는 커뮤니티 및 엔터프라이즈 오퍼레이터를 손쉽게 찾을 수 있습니다. OperatorHub는 모든 OpenShift 클러스터의 사이드바에서 찾을 수 있습니다.

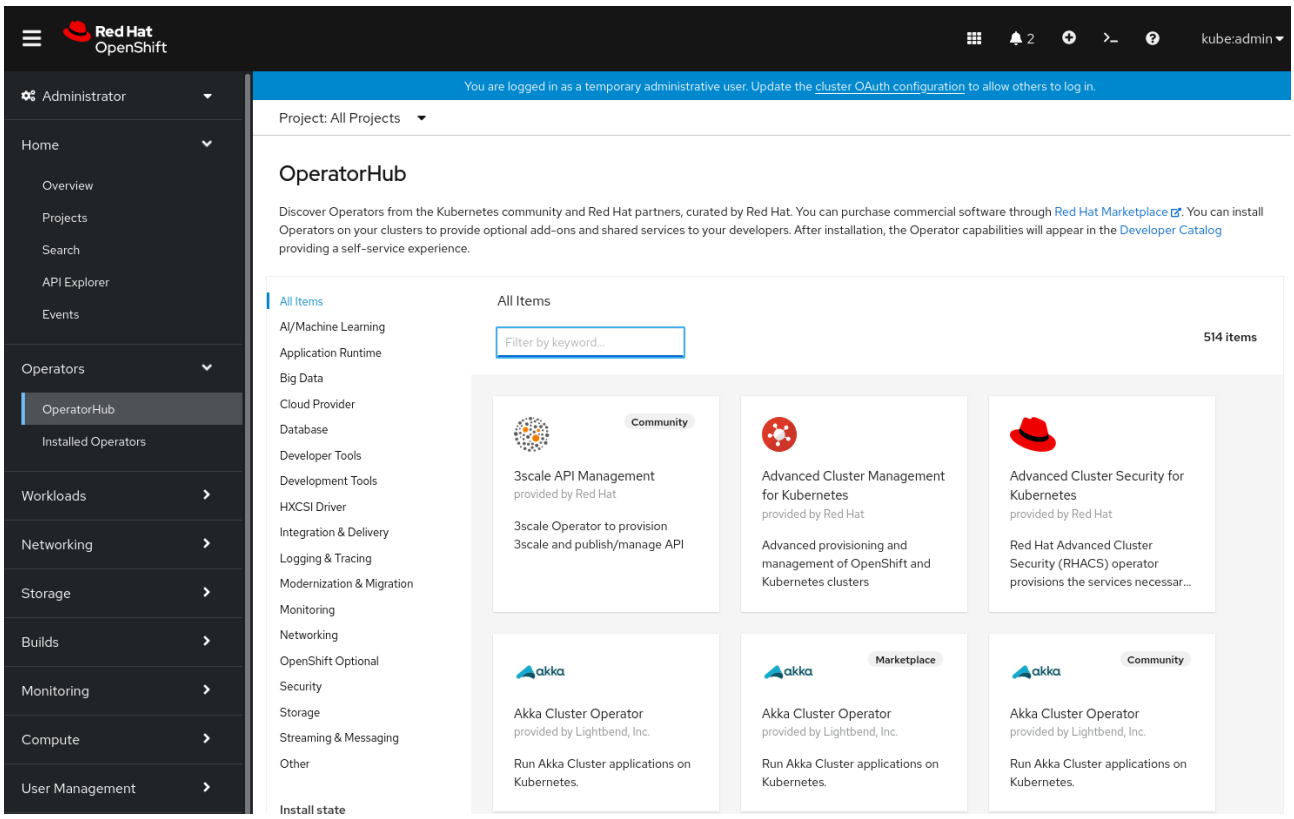


그림 6.2: OperatorHub

관리자는 오퍼레이터를 사용해 쿠버네티스 구성 및 코드에 대한 염려 없이 널리 사용되는 소프트웨어를 빠르고 쉽게 배포하고 유지 관리할 수 있습니다. 현재 Advanced Cluster Management, Red Hat OpenShift Service Mesh, Red Hat OpenShift Pipelines 등 많은 Red Hat 제품이 오퍼레이터로 패키징되어 있습니다. 하지만 MariaDB 데이터베이스, Couchbase 또는 IBM 블록 스토리지 드라이버와 같이 Red Hat 이외의 소프트웨어를 위한 방대한 오퍼레이터 라이브러리도 있습니다.

오퍼레이터에 관한 자세한 정보는 다음 링크를 참조하시기 바랍니다.

- Operatorhub.io
- OpenShift의 오퍼레이터

로깅에 대한 이해

Azure Red Hat OpenShift는 Red Hat OpenShift와 동일한 로깅 및 모니터링 아키텍처를 사용합니다. 두 제공 사항 모두 로깅을 위해 동일한 오퍼레이터를 사용합니다.

로그는 다음 세 가지 범주로 구분됩니다.

- **애플리케이션:** 인프라 컨테이너 애플리케이션 외에 클러스터에서 실행되는 사용자 애플리케이션이 생성하는 컨테이너 로그
- **인프라:** 클러스터와 OpenShift Container Platform 노드에서 실행되는 인프라 구성 요소가 생성하는 로그(예: 저널 로그). 인프라 구성 요소는 openshift*, kube* 또는 기본 프로젝트에서 실행되는 포드입니다.
- **감사:** 노드 감사 시스템인 auditd가 생성하는 로그(/var/log/audit/audit.log 파일에 저장됨)와 쿠버네티스 API 서버 및 OpenShift API 서버의 감사 로그

OpenShift 로깅에 대한 더 자세한 설명은 제품 도큐멘테이션의 Red Hat OpenShift 로깅에 대한 이해를 참조하세요.

클러스터 로그를 Azure Monitor로 전달하는 기능 사용하기

일반적인 통합은 Azure Red Hat OpenShift 로그를 Azure Monitor의 컨테이너 인사이트 서비스로 보내는 것입니다. 때로 이를 가리켜 Azure Log Analytics라고 합니다. 이렇게 하면 Azure에서 퍼시스턴트 저비용 로그 유지가 가능합니다.

OpenShift 로깅 아키텍처는 각 노드에서 Fluent Bit를 실행하고, 오퍼레이터가 취할 수 있는 첫 번째 접근 방식은 이 Fluent Bit 서비스의 구성을 조정하여 로그를 직접 전송하는 것입니다. 하지만 Azure Red Hat OpenShift에서 로깅 구성을 조정하는 것은 지원 정책의 범위에 속하지 않습니다. OpenShift에는 로그 전달을 위한 `ClusterLogForwarder`라는 빌트인 매커니즘이 여전히 지원됩니다.

`ClusterLogForwarder`를 사용하면 Elasticsearch, Flutentd, syslog에 로그를 전달할 수 있습니다. 하지만 Azure Monitor는 이러한 프로토콜 중 어느 것도 직접 지원하지 않습니다. 대안은 OpenShift의 로그를 수신하여 Azure Monitor로 전달하는 부가적인 중간 Fluent Bit 인스턴스를 마련하는 것입니다. 이 접근 방식에 관해서는 현재 Microsoft 도큐멘테이션과 커뮤니티 도큐멘테이션에 기술되어 있습니다.

모니터링에 대한 이해

관리형 서비스로서 요금을 지불하는 서비스의 일부로 제공되기 때문에 Azure Red Hat OpenShift에 대해 복잡한 사용자 정의 모니터링을 구현할 필요가 없어야 합니다.

하지만 Azure 컨테이너 인사이트를 사용해 OpenShift 컨테이너를 모니터링하길 원하는 경우가 매우 흔합니다. 이것은 현재 공개 미리 보기로 제공되는 기능으로서, 이 도큐멘테이션에 기술되어 있습니다.

업그레이드 및 패치

Azure Red Hat OpenShift 클러스터를 프로비저닝할 때 업데이트 채널은 선택되지 않습니다. 따라서 클러스터는 기본적으로 업데이트 수신을 시작하지 않습니다.

업데이트 채널을 보려면 탐색 사이드바에서 **관리** → **클러스터 설정**으로 이동하세요. Azure Red Hat OpenShift 클러스터를 프로비저닝한 직후의 클러스터 설정은 다음과 같습니다.

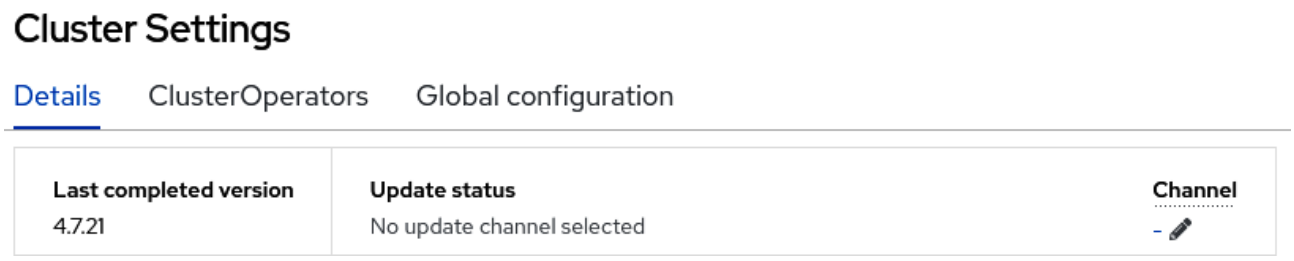


그림 6.3: 클러스터 프로비저닝 완료 후 클러스터 설정

업데이트 채널을 선택하려면 "-" 링크를 선택하세요. 사용 가능한 옵션은 다음과 같습니다.

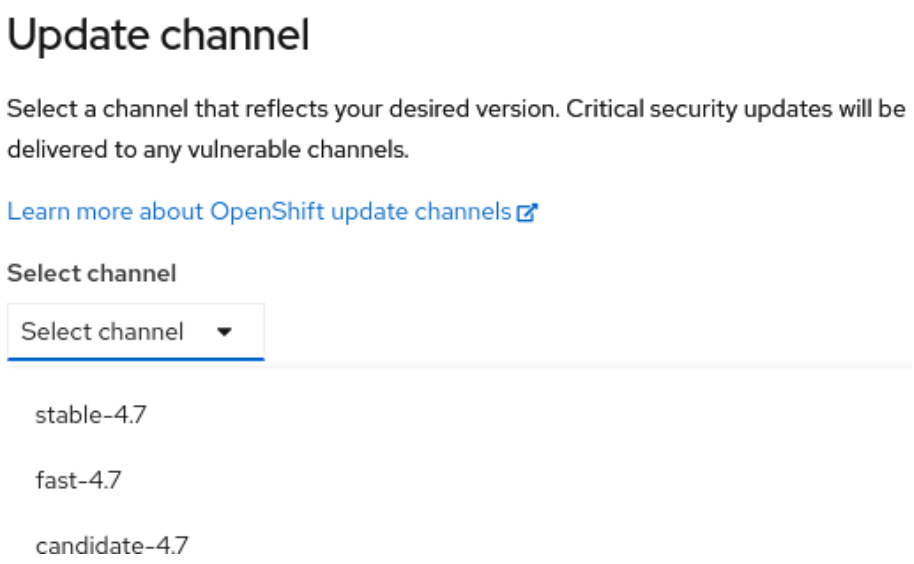


그림 6.4: 업데이트 채널 선택

`stable`, `fast`, `candidate`의 차이를 이해하려면 [채널 및 릴리스 업그레이드] 도큐멘테이션 페이지를 참조하세요.

프로덕션 단계의 모든 클러스터가 **stable** 채널을 실행할 것을 적극 권장합니다.

지원이 제공되는 버전의 라이프사이클

지원받는 Azure Red Hat OpenShift 버전이 어느 것인지 아는 것은 프로덕션 배포 계획을 수립하는 데 중요합니다. 공식 라이프사이클 페이지를 보시면 조직이 지원되는 버전과 지원되지 않는 버전이 어느 것인지 아는 데 도움이 되는 정보가 있습니다.

[Azure Red Hat OpenShift 지원 라이프사이클 페이지]

이 페이지에 포함된 정보를 발췌하여 간단히 정리하면 다음과 같습니다.

Azure Red Hat OpenShift는 다음과 같이 Red Hat OpenShift Container Platform의 두 가지 **GA(Generally Available)** 마이너 버전을 지원합니다.

- Azure Red Hat OpenShift에서 릴리스되는 최신 GA 마이너 버전(N이라고 하겠음)
- 이전의 마이너 버전 하나(N-1)

Red Hat OpenShift Container Platform은 의미에 따른 버전 관리를 사용합니다. 의미에 따른 버전 관리는 여러 수준의 버전 번호를 사용해 여러 수준의 버전 관리를 지정합니다. 다음 표에는 의미에 따른 버전 번호의 여러 부분이 나와 있는데, 이 경우 예시 버전 번호 4.9.3을 사용하고 있습니다.

메이저 버전 (x)	마이너 버전 (y)	패치 (z)
4	9	3

이 버전의 각 번호는 이전 버전과의 일반적인 호환성을 나타냅니다.

- **메이저 버전:** 이때 메이저 버전 릴리스에 대한 계획은 없습니다. 메이저 버전은 비호환 API 변경 사항 또는 역호환성이 중단될 수 있는 경우 변경됩니다.
- **마이너 버전:** 약 3개월마다 출시됩니다. 마이너 버전 업그레이드에는 기능 추가, 개선 사항, 사용 중단, 제거, 버그 수정, 보안 개선 사항이 포함될 수 있습니다.
- **패치:** 일반적으로 매주 또는 필요에 따라 릴리스됩니다. 패치 버전 업그레이드에는 버그 수정 및 보안 개선 사항이 포함될 수 있습니다.

지원되는 버전에 대해 더 자세히 알고 싶다면 Azure Red Hat OpenShift 지원 라이프사이클 페이지를 꼼꼼히 읽어보세요.

클러스터 확장

Red Hat OpenShift Container Platform, 그리고 더 나아가 Azure Red Hat OpenShift는 확장 가능한 아키텍처를 염두에 두고 구축되었습니다. OpenShift의 맥락에서 확장 계획을 수립할 때 일반적으로 관리자와 애플리케이션 소유자는 클러스터 확장과 애플리케이션 확장을 서로 뚜렷이 구분되는 두 가지 주제로 간주해야 합니다.

이 섹션에서는 클러스터 확장에 대해 다루며 애플리케이션 확장을 별도의 섹션으로 다루는 간략한 참고 사항이 있습니다.

최대 지원 개수

클러스터 확장은 추가 작업자 노드를 클러스터에 추가하는 것을 가리킵니다. 그러면 클러스터는 클러스터에서 실행되는 애플리케이션에 추가 컴퓨팅 용량을 제공합니다. 어떤 경우에 컨트롤 플레인을 확장해야 하는지에 관한 의문도 자연스럽게 제기됩니다. Azure Red Hat OpenShift는 Azure Red Hat OpenShift 클러스터에서 지원되는 작업자 노드의 최대 수(현재 60개)까지 확장하기에 충분한 용량과 함께 원칙상 제공되는 세 개의 컨트롤 플레인 노드를 배포합니다.

이 문서를 작성하는 시점에 지원되는 컨트롤 플레인 노드의 인스턴스 크기는 `Standard_D8s_v3`, `Standard_D16s_v3`, `Standard_D16s_v3` 세 가지입니다.

작업자 노드에 대해 지원되는 추가 Azure 인스턴스 유형으로 컴퓨팅 최적화(F 시리즈), 메모리 최적화(E 시리즈), 범용(D 시리즈)이 있습니다.

Azure Red Hat OpenShift에 대해 현재 지원되는 Azure 인스턴스 유형의 목록은 지원 정책 페이지에서 보실 수 있습니다.

최소 배포와 제로로 축소

어떤 경우 조직은 테스트 및 개발 목적으로, 또는 가용성 축소가 용인되는 다른 활용 사례를 위해 더 작은 규모의 클러스터를 배포하길 원할 수 있습니다. 현재 최소 클러스터 크기는 컨트롤 플레인 노드 3개, 애플리케이션 노드 3개이며, 이보다 작게 축소하는 것은 지원하지 않습니다.

클러스터를 수동으로 확장

Azure Portal을 살펴본 오퍼레이터는 Azure 가상 머신을 직접 만들어 부가적인 작업자 노드를 Azure Red Hat OpenShift 클러스터에 추가하고 싶은 유혹을 느낄 수 있습니다. 하지만 Azure 관리자의 관점에서 볼 때 Azure Red Hat OpenShift를 포함하는 리소스 그룹이 "잠긴 상태"이기 때문에 이것은 불가능합니다. 이 경우는 권한과 관계가 없습니다. **소유자** 권한이 있는 사용자라도 Azure Red Hat OpenShift 리소스 그룹의 콘텐츠를 수정할 수 없습니다. 따라서 Azure Red Hat OpenShift 리소스 그룹 내에서 수동으로 가상 머신을 만들려고 하는 경우 '권한 거부' 오류가 발생하게 됩니다.

Azure Red Hat OpenShift 확장에 의도된 역학은 OpenShift `MachineSets` 기능을 사용하는 것입니다. 이 기능으로 OpenShift는 지시가 있을 때 새로운 애플리케이션 노드를 배포합니다. 클러스터 관리자 권한이 있는 사용자는 **컴퓨팅**에서 **MachineSets**를 확인할 수 있습니다.

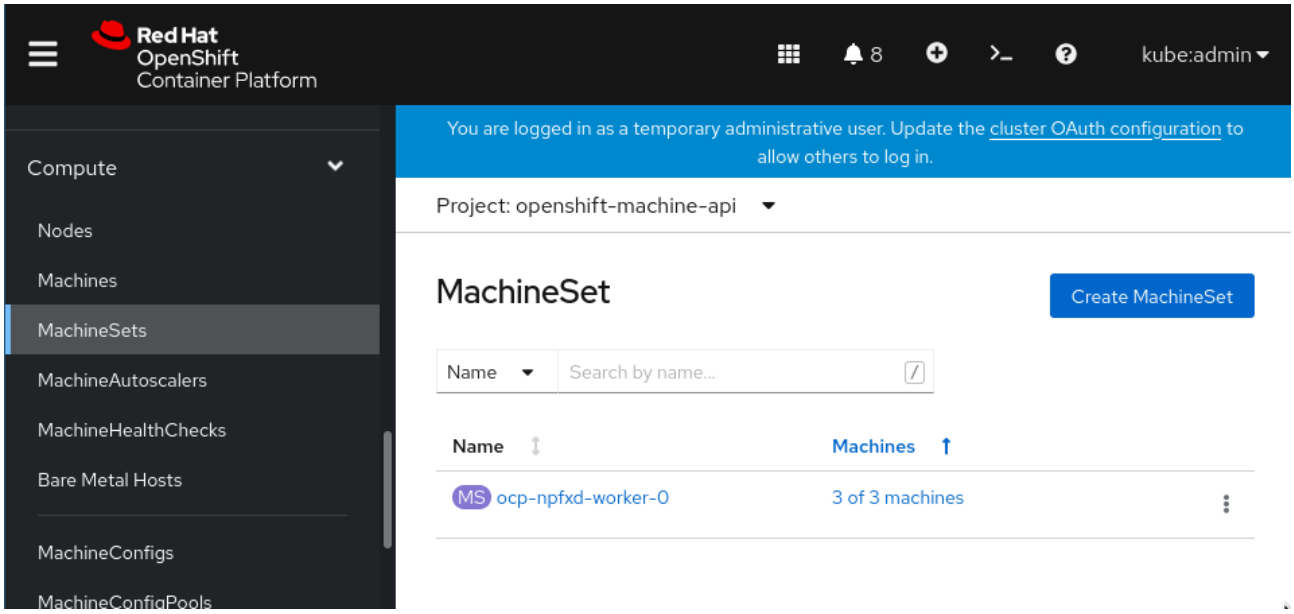


그림 6.5: MachineSets에 대한 관리자 액세스

`MachineSet` 애플리케이션 노드의 "편집" 메뉴를 클릭하면 **머신 개수 편집**을 선택하여 새로운 애플리케이션 노드 가상 머신을 간편하게 프로비저닝할 수 있음을 알 수 있습니다.

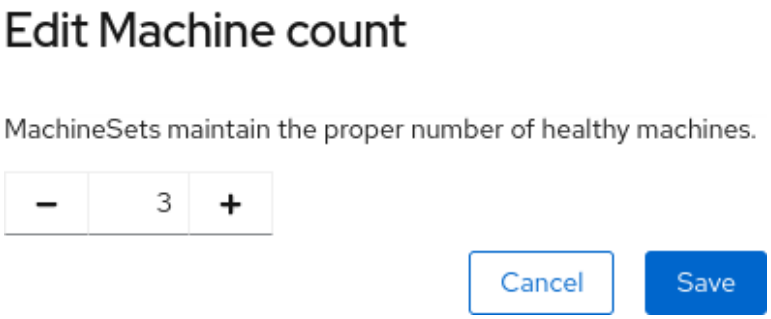


그림 6.6: 머신 개수 편집

개수가 업데이트되었으면 관리자는 Azure Portal 또는 **컴퓨팅** → **머신** 보기로 이동하여 새로운 애플리케이션 노드 가상 머신이 프로비저닝되고 있는지 확인할 수 있습니다.

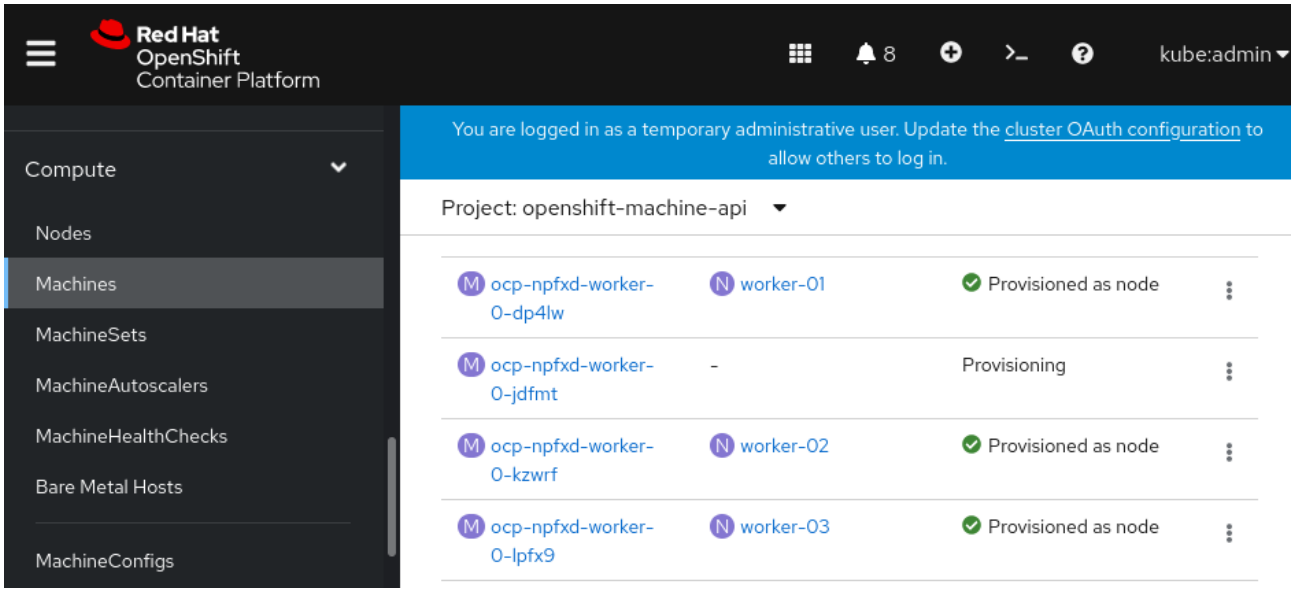


그림 6.7: 머신 보기

대부분의 Azure 지역에서는 추가 가상 머신을 프로비저닝하는 데 일반적으로 5분 정도 소요됩니다.

관리자는 이 새로운 용량을 온라인으로 전환하기 위해 프로비저닝 후 활동을 수행할 필요가 없습니다. OpenShift는 클러스터에 자동으로 제공되고, 애플리케이션은 필요 시 OpenShift를 사용합니다.

클러스터 자동 확장

Azure Red Hat OpenShift 기본 배포는 자동 확장 기능을 켠 상태로 구성되지 않지만 이 기능을 활성화하는 것은 간단합니다. 관리자는 Azure Red Hat OpenShift **컴퓨팅** 사이드바 메뉴에서 찾을 수 있는 `MachineAutoscaler` 리소스를 생성하기만 하면 됩니다.

MachineAutoscaler 리소스는 MachineSet에서 작동합니다. 그러면 MachineSet이 필요에 따라 애플리케이션 노드 가상 머신 용량을 생성하거나 삭제합니다. 다음 예시는 MachineSet에서 머신의 최소 또는 최대 개수를 유지 관리할 수 있음을 보여줍니다.

```
apiVersion: autoscaling.openshift.io/v1beta1
kind: MachineAutoscaler
metadata:
    name: worker-us-east-1a
    namespace: openshift-machine-api
spec:
    minReplicas: 1
    maxReplicas: 12
    scaleTargetRef:
        apiVersion: machine.openshift.io/v1beta1
        kind: MachineSet
        name: worker
```

OpenShift에도 특정한 양의 RAM, CPU 또는 이와 유사한 것을 가용 상태로 유지하는 것을 기반으로 확장을 가능케 하는 클러스터 자동 스케일러라는 개념이 있습니다. MachineAutoscaler와 ClusterAutoscaler 중에 어느 것을 선택하느냐는 클러스터에서 추가하고 제거하려는 용량이 얼마인지에 달려 있습니다.

MachineAutoscalers 및 ClusterAutoscalers에 관한 Red Hat OpenShift 도큐멘테이션

애플리케이션 확장

마이크로서비스 애플리케이션 확장은 이 책의 범위를 벗어나는 복잡한 주제입니다. 하지만 이 주제에 대해 두루 살펴볼 수 있는 두 가지 유용한 문서를 안내해 드립니다.

- HorizontalPodAutoscaler: 실행하고자 하는 포드의 최소 및 최대 개수뿐 아니라 포드가 목표로 삼아야 하는 CPU 활용도 또는 메모리 활용도까지도 지정합니다.
- Knative를 이용한 서버리스와 제로로 축소

클러스터는 실행 중인 컨테이너와 클러스터에 남은 용량을 모니터링합니다. Knative 또는 HorizontalPodAutoscaler가 클러스터에서 현재 사용 가능한 리소스보다 많은 리소스를 요청하는 경우 ClusterAutoscaler 또는 MachineSet이 확장되어 요청된 리소스를 클러스터에 추가해야 합니다.

이 접근 방식을 사용해 애플리케이션과 클러스터 자체를 수요에 따라 동시에 확장 및 축소할 수 있습니다.

풀(pull) 암호 설정(OpenShift Cluster Manager에 등록)

Azure Red Hat OpenShift 신규 배포의 경우 "풀(pull) 암호"가 `cloud.redhat.com`에 구성되어 있지 않습니다. 따라서 클러스터가 Red Hat 하이브리드 클라우드 콘솔(http://console.redhat.com)에 기본적으로 표시되지는 않습니다. 이 콘솔을 OpenShift Cluster Manager라고 합니다.

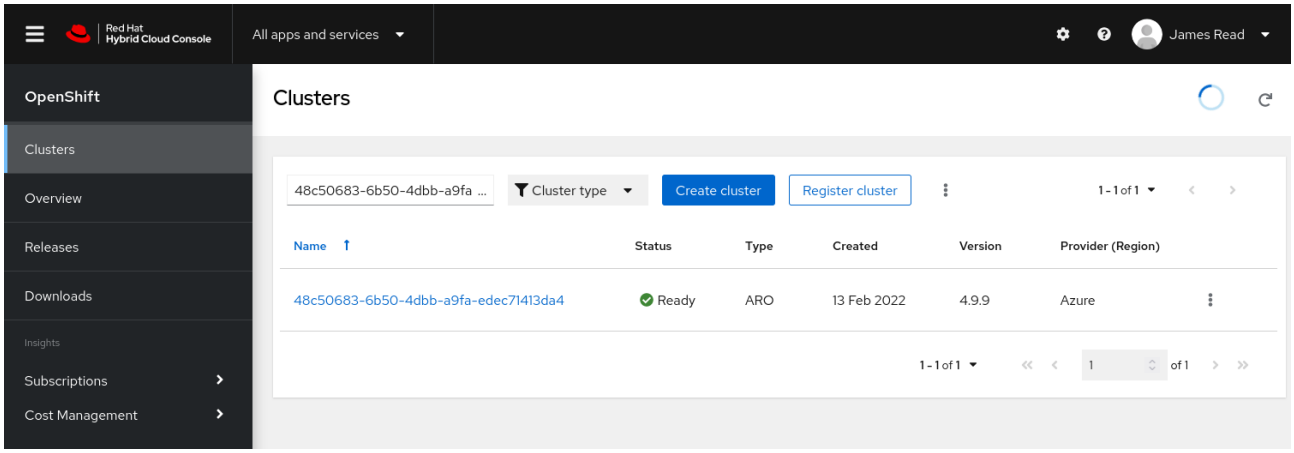


그림 6.8: Azure Red Hat OpenShift 클러스터를 보여주는 OpenShift Cluster Manager

`cloud.redhat.com`에 풀(pull) 암호를 구성하는 것은 매우 간단합니다. http://console.redhat.com의 OpenShift Cluster Manager 포털에 표시할 수 있을 뿐 아니라 스탠다드 지원 티케팅 시스템을 통해 Red Hat에 직접 지원 티켓을 제출할 수도 있습니다.

풀(pull) 암호 설정 방법에 관한 지침은 다음과 같습니다.

- 풀(pull) 암호를 추가하거나 업데이트하는 방법

풀(pull) 암호를 설정함으로 얻을 수 있는 부가적 이점은 고객이 Red Hat에 직접 지원 티켓을 제출할 수 있다는 것입니다. 풀(pull) 암호는 클러스터가 지원 담당 직원에게 보이도록 허용하는 Red Hat 계정에 관한 권한을 부여합니다. Azure Red Hat OpenShift에 대한 지원 티켓을 여는 것은 다른 Red Hat 제품과 동일합니다.

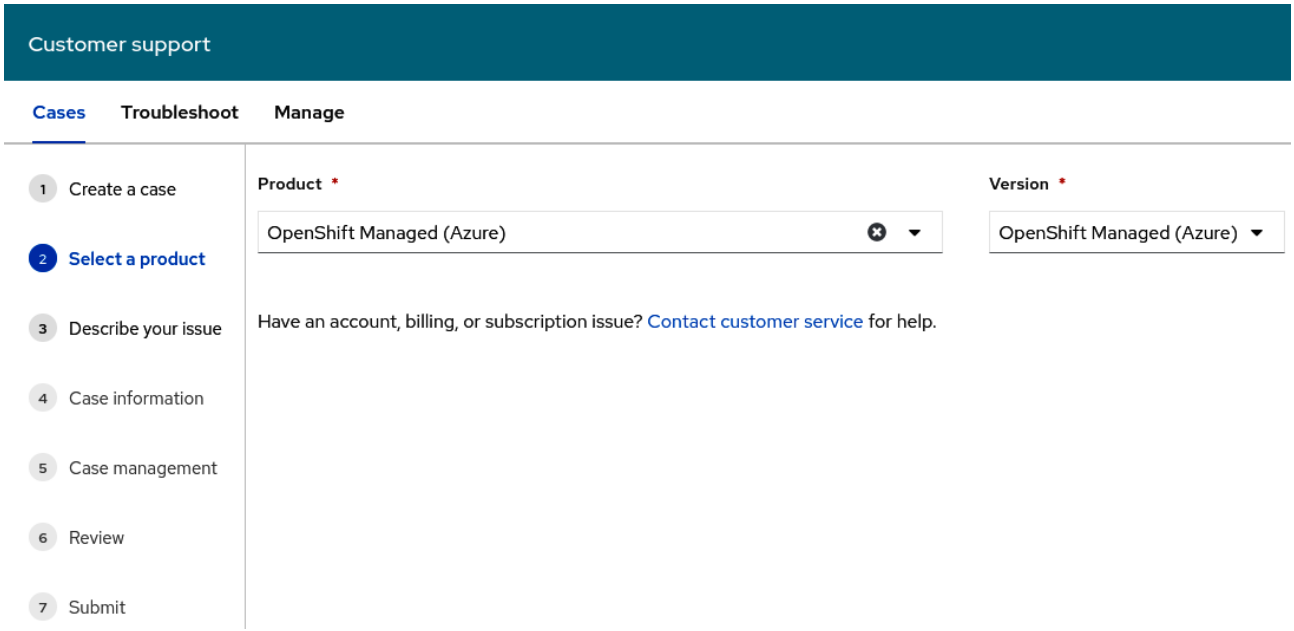


그림 6.9: 지원 사례 생성

제한 범위

개발 또는 애플리케이션 팀이 여러 개의 컨테이너화된 애플리케이션을 배포하고 나서 얼마 지나지 않아 한 애플리케이션이 "오작동"하여 클러스터에서 불필요한 리소스를 소비하기 시작합니다. 일례로 메모리 유출이 발생하는 결함 있는 애플리케이션 또는 100%가 아닌 10%의 CPU만 소비해도 확장되도록 잘못 설정된 애플리케이션이 있을 수도 있습니다. 이러한 오작동 애플리케이션이 제어할 수 없는 크기로 확장되고 너무 많은 리소스를 소비하는 것을 방지하려면 `LimitRange`를 사용하는 것이 좋습니다.

`LimitRange`를 통해 프로젝트의 특정 오브젝트에 대한 리소스 소비를 제한할 수 있습니다. `LimitRange`를 적용할 수 있는 대상은 다음과 같습니다.

- 포드 및 컨테이너: 포드와 포드의 컨테이너에 사용할 CPU 및 메모리에 대한 최소 및 최대 요구 사항을 설정할 수 있습니다.
- 이미지 스트림: ImageStream 오브젝트에서 이미지 및 태그의 수에 대한 제한을 설정할 수 있습니다.
- 이미지: 내부 레지스트리에 푸시할 수 있는 이미지의 크기를 제한할 수 있습니다.
- **퍼시스턴트 볼륨 클레임(PVC):** 요청할 수 있는 PVC의 크기를 제한할 수 있습니다.

컨테이너에 적용되어 허용되는 최소, 최대 및 기본 CPU 및 메모리 요청을 제한하는 제한 범위의 한 가지 예는 다음과 같습니다.

```
apiVersion: “v1”
kind: “LimitRange”
metadata:
  name: “resource-limits”
spec:
  limits:
    - type: “Container”
      max:
        cpu: “2”
        memory: “1Gi”
      min:
        cpu: “100m”
        memory: “4Mi”
      default:
        cpu: “300m”
        memory: “200Mi”
      defaultRequest:
        cpu: “200m”
        memory: “100Mi”
      maxLimitRequestRatio:
        cpu: “10”
```

이 `LimitRange` 코드 스니펫은 YAML을 복사하여 Red Hat OpenShift 콘솔의 편집기에 직접 붙여넣거나 `oc apply -f limitrange.yaml`이 포함된 파일에서 적용할 수 있습니다.

제한 범위 도큐멘테이션

퍼시스턴트 스토리지

Azure Red Hat OpenShift로 배포되는 가상 머신은 Red Hat CoreOS를 설치하고 Azure Red Hat OpenShift 서비스를 실행할 목적으로 Azure 디스크를 첨부한 상태로 제공됩니다. 이 디스크는 애플리케이션이 사용해서는 안 되며 OpenShift 클러스터 자체에만 사용해야 합니다.

퍼시스턴트 스토리지가 필요한 애플리케이션은 쿠버네티스 `PersistentVolume` 기능을 사용해야 하며, 이 개념을 자세히 설명한 퍼시스턴트 스토리지에 대한 이해라는 OpenShift 도큐멘테이션에 탁월하게 기술되어 있습니다. Azure Red Hat OpenShift는 기술적으로 모든 `PersistentVolume` 제공업체를 자체 관리형 OpenShift 설치로 지원하는 반면, Azure에서 가장 흔히 사용되는 퍼시스턴트 스토리지는 다음과 같습니다.

이름	유형	액세스 모드	스토리지 클래스
Azure Files	파일 시스템, POSIX 이외 규격	ReadWriteOnce	Azure File
Azure Disk	블록	ReadWriteOnce	Azure Disk
Red Hat OpenShift Data Foundation	파일 시스템, 블록, 오브젝트	(몇 가지)	OCS/ODF 문서

보안 및 컴플라이언스

현재 Red Hat OpenShift 도큐멘테이션에는 다수의 보안 제어 기능을 구현하고 컴플라이언스를 유지 관리하는 방법의 이해에 관한 상당한 양의 콘텐츠가 포함되어 있습니다.

OpenShift 보안 및 컴플라이언스 도큐멘테이션

도큐멘테이션의 관련 섹션은 다음과 같습니다.

- 컨테이너 보안이 OpenShift에서 작동하는 방식에 대한 자세한 설명. OpenShift는 다른 쿠버네티스 기반 서비스에서 찾을 수 없는 다수의 즉시 사용 가능한 컨테이너용 보안 기능을 제공하며, 이러한 기능은 조직과 애플리케이션을 안전하게 유지하는 데 도움이 된다는 점을 이해하는 것이 중요합니다.
- 포드 취약점 검사
- 감사 로그 액세스
- 인증서 구성

여기에는 다음과 같은 몇 가지 유용한 보안 관련 오퍼레이터도 포함됩니다.

- **컴플라이언스 오퍼레이터:** 검사를 실행하고 일반적인 보안 문제 해결을 위한 권장 사항을 제공합니다.
- **파일 무결성 확인:** 파일, 특히 민감한 보안 구성 파일이 변경되지 않았는지 지속적으로 확인합니다.

요약

이 장에서는 조직이 새로 배포한 클러스터를 프로덕션 애플리케이션에 적합하게 만드는 과정에서 일반적으로 고려하는 대부분의 태스크와 주제에 대해 알아보았습니다. 배포할 때마다 이러한 주제를 일일이 살펴볼 필요는 없지만, 첫 번째 클러스터를 배포할 때는 퍼시스턴트 스토리지, 제한, 인증을 비롯하여 이 장에서 언급한 기타 여러 주제를 고려해야 합니다.

일반적으로 조직은 최대한 많은 프로비저닝 후 태스크를 자동화하려고 노력할 것입니다. 예를 들어 Azure Active Directory 설정 및 구성은 PowerShell 스크립트 또는 몇 가지 ARM 템플릿을 사용해 거의 완료할 수 있습니다. 다수의 클러스터를 배포하는 경우 이렇게 하면 반복적인 태스크에 소요되는 시간을 단축할 수 있습니다.

이 책의 다음 장에서는 프로덕션 레디 Azure Red Hat OpenShift 클러스터 위에 샘플 애플리케이션을 배포하는 것에 관해 기술합니다.

7장

샘플 애플리케이션 배포

이 가이드는 Azure Red Hat OpenShift로 프로덕션 단계에 진입하는 데 필요한 것이 무엇인지 알아보고자 하는 대상 독자인 기술 인력(개발자 및 운영 담당자)에 도움이 되는 내용으로 주로 구성되어 있습니다. 이 가이드는 독자에게 Red Hat OpenShift에 대한 기본적 이해가 있다고 가정합니다. Red Hat OpenShift의 아키텍처, 관리 포털, 사용자 환경이 Azure Red Hat OpenShift와 동일하기 때문입니다.

이 장은 "과일 스무디" 애플리케이션이라고 하는 단순한 샘플 애플리케이션을 배포하는 방법에 관한 짧고 간단한 설명 자료의 역할을 합니다. 이 애플리케이션은 Azure Red Hat OpenShift 사용법에 대한 이해를 증진하는 빠른 시작 및 알림이 역할을 해야 합니다. 이것은 Azure Kubernetes Service용으로 유지 관리되는 샘플 애플리케이션이며 Azure Red Hat OpenShift 위에 이 애플리케이션을 배포하는 작업은 OpenShift가 쿠버네티스와 100% 호환된다는 증거를 제공하기 위해 수행됩니다.

이 장은 대체로 http://aroworkshop.io의 내용을 바탕으로 하고 있으며 여기에 설명과 맥락을 추가했습니다.

평가 애플리케이션 개요

애플리케이션 아키텍처는 다음과 같이 단순합니다.

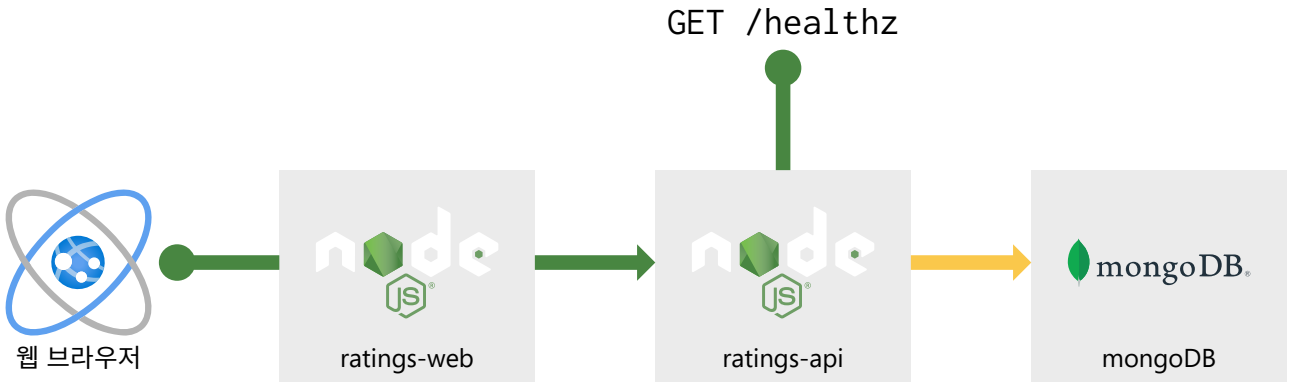


그림 7.1: 과일 스무디 애플리케이션의 아키텍처

위 다이어그램에서 애플리케이션이 세 가지 서비스와 두 개의 퍼블릭 엔드포인트로 구성되어 있음을 알 수 있습니다. 이를 정리하면 아래 표와 같습니다. 다이어그램 왼쪽의 첫 번째 엔드포인트는 퍼블릭 HTML 웹 애플리케이션을 보는 사용자의 웹 브라우저를 나타낸 것입니다. 두 번째 엔드포인트인 `healthz`는 `ratings-api` 서비스의 상태 점검입니다. 개별 셀프 서비스에 대한 설명과 각 서비스의 리포지토리로 연결되는 링크는 다음 표에서 보실 수 있습니다.

구성 요소	설명	GitHub 리포지토리
`rating-web`	공용 웹 프론트엔드 — "웹사이트"	GitHub 리포지토리
`rating-api`	이 서비스는 웹 UI에서 입력을 받아 데이터베이스에 저장합니다. 또한 데이터베이스의 결과를 포트 3000의 웹 애플리케이션으로 다시 제공합니다.	GitHub 리포지토리
`mongodb`	미리 로드된 데이터가 포함된 NoSQL 데이터베이스	데이터

GitHub 리포지토리 링크를 방문하시면 이러한 애플리케이션에 대해 더 자세히 살펴보고 이해할 수 있습니다. 이어지는 지침에서는 이러한 애플리케이션을 각기 배포하는 방법을 단계별로 안내합니다.

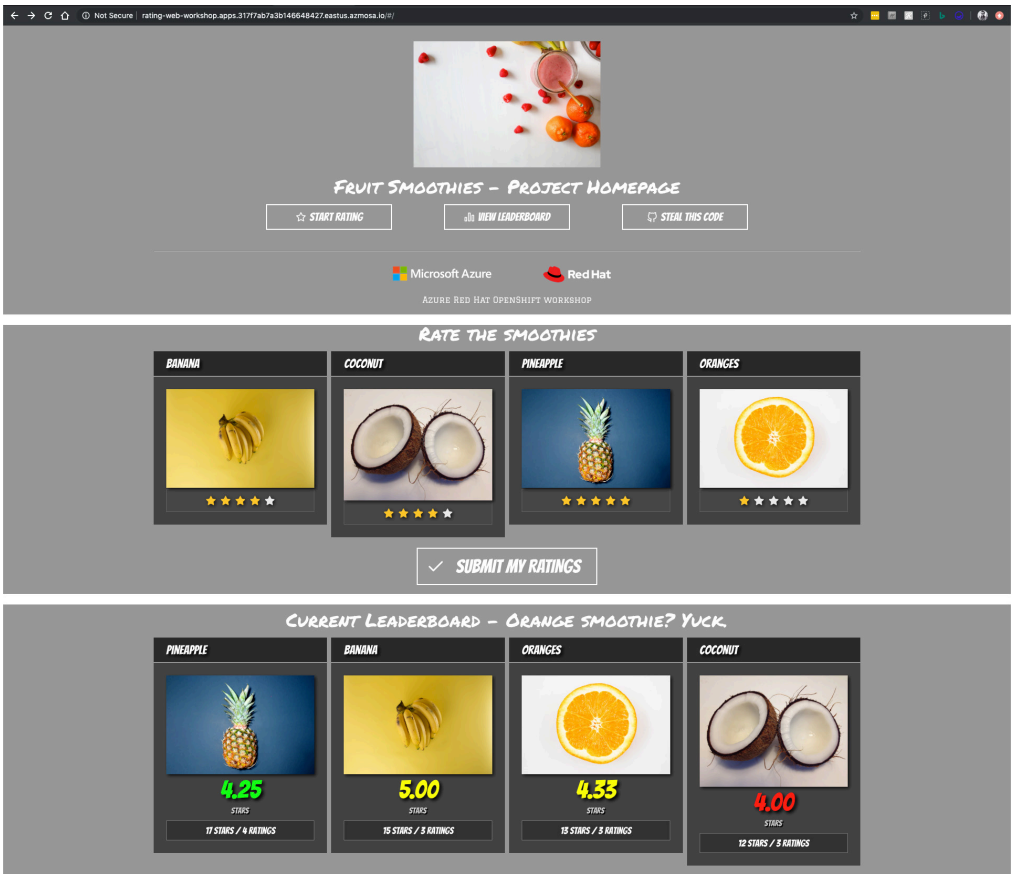


그림 7.2: 과일 스무디 애플리케이션의 개요를 보여주는 스크린샷

완료하고 나면 위 스크린샷과 비슷한 웹 애플리케이션이 실행됩니다. 개발자와 운영 팀이 애플리케이션을 Azure Red Hat OpenShift에 배포하는 방식에 대해서도 더 깊이 이해해야 합니다. 그러면 애플리케이션을 Azure Red Hat OpenShift에 직접 배포할 때 의사 결정하는 과정에서 도움이 됩니다.

클러스터 생성 및 연결

이 장의 나머지 부분에서는 독자에게 작업용 Azure Red Hat OpenShift 환경이 있다고 가정합니다. 이 평가 애플리케이션에는 특별한 요구 사항이 없으므로 이 애플리케이션은 비어 있는 새로운 Azure Red Hat OpenShift 환경에 배포됩니다. 클러스터를 아직 프로비저닝하지 않았다면 다음 장을 참조하세요.

- *4장: 프로비저닝 전 – 엔터프라이즈 아키텍처에 관한 질문*
- *5장: Azure Red Hat OpenShift 클러스터 프로비저닝*

*5장: Azure Red Hat OpenShift 클러스터 프로비저닝*의 *클러스터에 액세스* 섹션은 이전에 프로비저닝했을 수 있는 클러스터에 액세스하는 방법을 알려주는 유용한 자료입니다.

웹 콘솔에 로그인

각 Azure Red Hat OpenShift 클러스터는 OpenShift 웹 콘솔용 DNS 주소가 있습니다. `az aro list` 커맨드를 사용해 현재 Azure 서브스크립션의 클러스터를 나열할 수 있습니다.

```
az aro list -o table
```

클러스터 웹 콘솔의 URL이 나열됩니다. 새 브라우저 탭에서 이 링크를 열고 `kubeadmin` 사용자, 또는 프로젝트 생성 권한이 있는 다른 사용자 계정으로 로그인합니다.

로그인하면 Azure Red Hat OpenShift 웹 콘솔이 표시되어야 합니다.

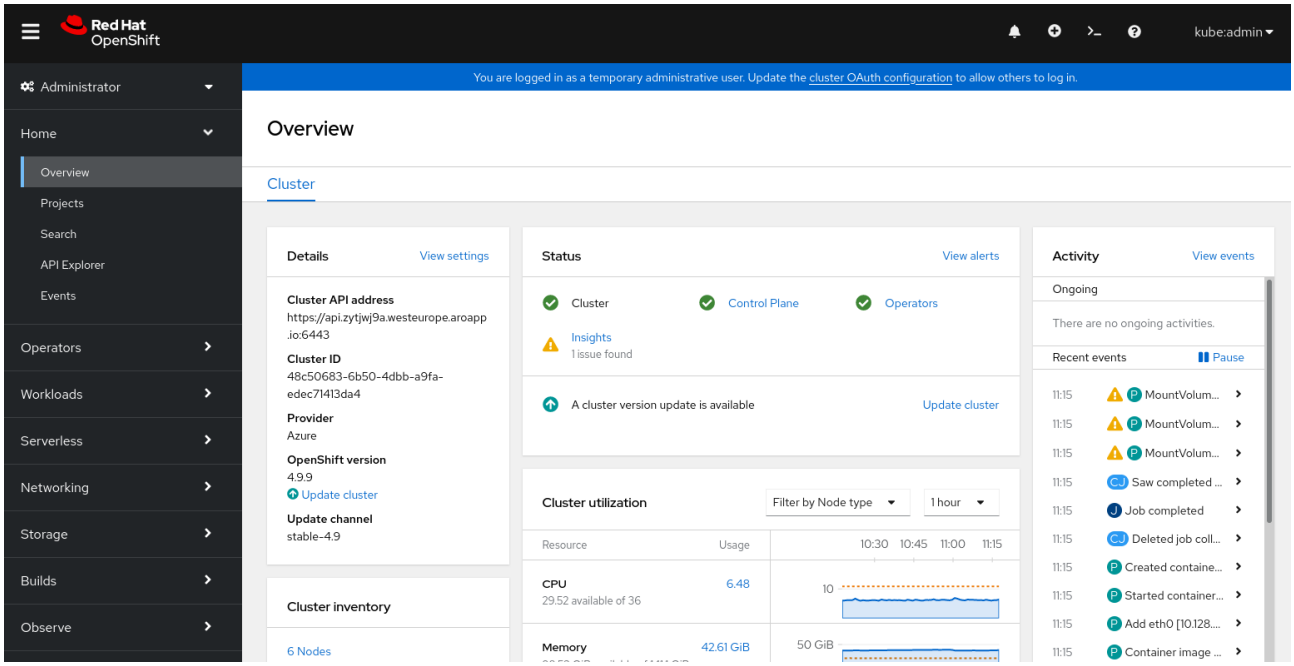


그림 7.3: Azure Red Hat OpenShift 웹 콘솔

OpenShift클라이언트 설치

Azure Cloud Shell을 열거나 로컬 Linux 터미널을 사용하여 OpenShift 커맨드라인 클라이언트를 설치합니다. 이는 커맨드라인에서 클러스터에 액세스하는 데 필요합니다. 이 작업에 관한 지침은 다음과 같습니다.

```
cd ~
curl https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-linux.tar.gz > openshift-client-linux.tar.gz

mkdir openshift

tar -zxvf openshift-client-linux.tar.gz -C openshift

echo 'export PATH=$PATH:~/openshift' >> ~/.bashrc && source ~/.bashrc
```

또는 Windows나 Mac의 경우 다운로드 링크는 다음과 같습니다.

- https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-windows.zip
- https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-mac.tar.gz

다운로드한 패키지 중 하나에서 oc 커맨드를 실행할 수 있어야 합니다.

로그인 커맨드 및 토큰 검색

클라이언트가 설치된 후 클러스터에 로그인하려면 토큰을 가져와야 합니다. OpenShift 웹 콘솔에 로그인하고 우측 상단의 사용자 이름을 클릭한 후 **로그인 커맨드 복사**를 클릭합니다.

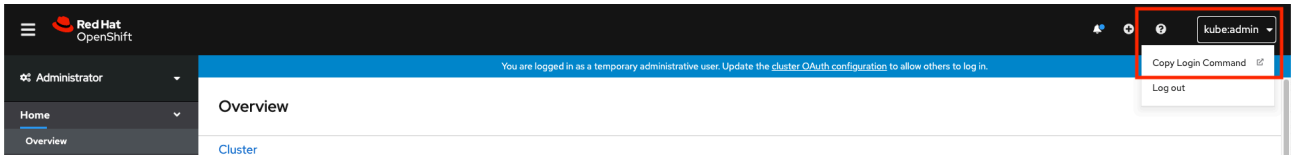


그림 7.4: 로그인 커맨드 복사를 클릭하여 클러스터에 로그인하기

셸에 로그인 커맨드를 붙여넣습니다(로컬 Linux 터미널 또는 Azure Cloud Shell). 클러스터에 연결할 수 있어야 합니다.

그림 7.5: 로그인 커맨드를 사용해 클러스터에 연결하기

연결되면 프로젝트를 생성하는 단계로 나아갈 수 있습니다.

프로젝트 생성

OpenShift의 프로젝트는 평가 애플리케이션이 포함된 논리적 폴더와 같습니다. Red Hat OpenShift에서는 모든 컨테이너와 애플리케이션이 어떤 종류의 프로젝트 내부에 있어야 합니다. 프로젝트를 사용해 애플리케이션, 심지어 부서까지도 분리할 수 있습니다. 다음 몇 가지 지침에서는 프로젝트 생성 방법을 설명합니다.

웹 인터페이스에서 프로젝트를 생성할 수 있지만 이 지침에서는 다음과 같은 커맨드라인을 사용합니다.

```
oc new-project workshop
```

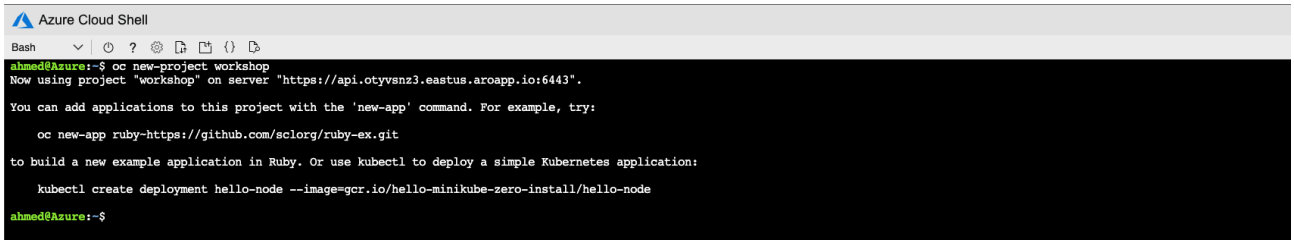


그림 7.6: Azure Cloud Shell에서 새로운 워크숍 생성하기

프로젝트가 생성된 후에는 `oc project workshop`을 사용하여 프로젝트로 전환할 수 있습니다. 다음 단계는 이 장을 시작할 때 소개한 세 개의 마이크로서비스 중 첫 번째인 MongoDB를 배포하는 것입니다. MongoDB는 방금 생성한 프로젝트에 배포됩니다.

리소스

- Azure Red Hat OpenShift 도큐멘테이션 – CLI 시작하기
- Azure Red Hat OpenShift 도큐멘테이션 – 프로젝트

배포 MongoDB

Azure Red Hat OpenShift는 새로운 MongoDB 데이터베이스 서비스를 손쉽게 생성할 수 있도록 컨테이너 이미지 및 템플릿을 제공합니다. 이 템플릿은 비밀번호 값 자동 생성을 비롯한 사전 정의된 기본값으로 모든 필수 환경 변수(사용자, 비밀번호, 데이터베이스 이름 등)를 정의할 수 있는 매개 변수 필드를 제공합니다. 또한 배포 구성 및 서비스를 정의합니다.

MongoDB 인스턴스는 Docker Hub에서 컨테이너 이미지로 직접 배포됩니다. OpenShift를 이용하면 웹 콘솔을 통해 이 모든 것을 수행할 수 있습니다. 메뉴 상단에서 개발자 관점으로 전환하여 **추가** 페이지로 이동한 다음, **컨테이너 이미지**를 선택합니다.

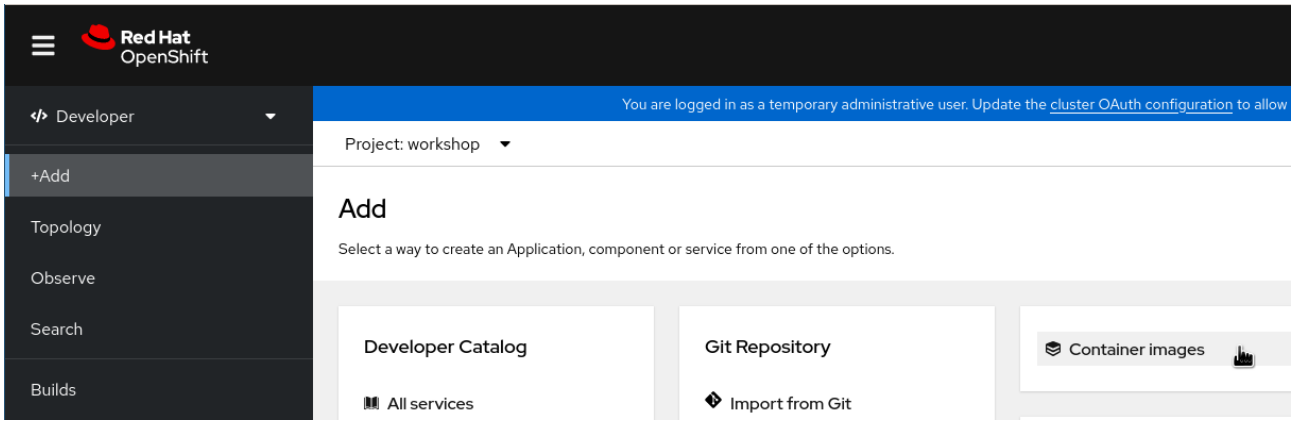


그림 7.7: 컨테이너 이미지 추가

그러면 **이미지 배포** 페이지로 이동합니다. 양식에 다음과 같이 입력합니다.

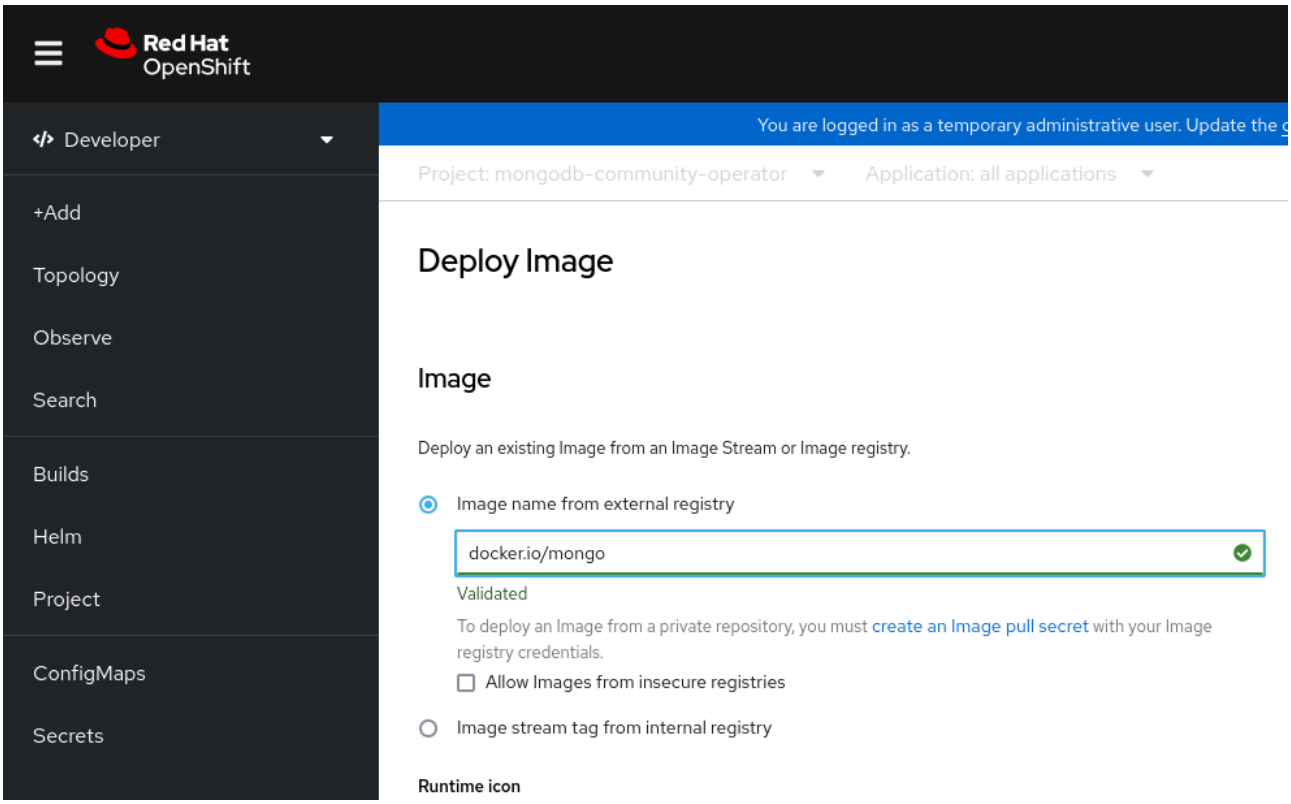


그림 7.8: 이미지 배포를 위해 양식 입력하기

양식에 반드시 다음과 같이 값을 설정해야 합니다.

필드	값
외부 레지스트리의 이미지 이름	`docker.io/mongo`
런타임 아이콘	`mongodb`
애플리케이션 이름	평가
이름	`mongodb`
애플리케이션 경로 생성	체크 표시 안 함 — 경로를 통해 외부에서 데이터베이스에 액세스할 수 있음, 이 애플리케이션에는 필요 없음
리소스 유형	배포

양식 하단에 이르면 환경 변수를 설정할 수 있도록 배포 링크를 선택하여 양식을 펼칩니다.

다음 환경 변수는 데이터베이스를 처음에 시작할 때 데이터베이스 초기화를 위한 기본값의 역할을 합니다.

```
Name                    Value
MONGO_INITDB_DATABASE   ratingsdb
```

MongoDB 데이터베이스에는 사용자와 비밀번호가 설정되지 않습니다. 이것이 기본 구성이며 인증은 비활성화됩니다.

환경 변수를 다시 확인한 다음, **생성** 버튼을 클릭하여 계속해서 MongoDB 컨테이너를 배포합니다.

잠시 후 MongoDB 인스턴스가 `workspace` 프로젝트 내에서 실행되어야 합니다. **토폴로지** 보기로 전환하면 이 배포를 볼 수 있습니다.

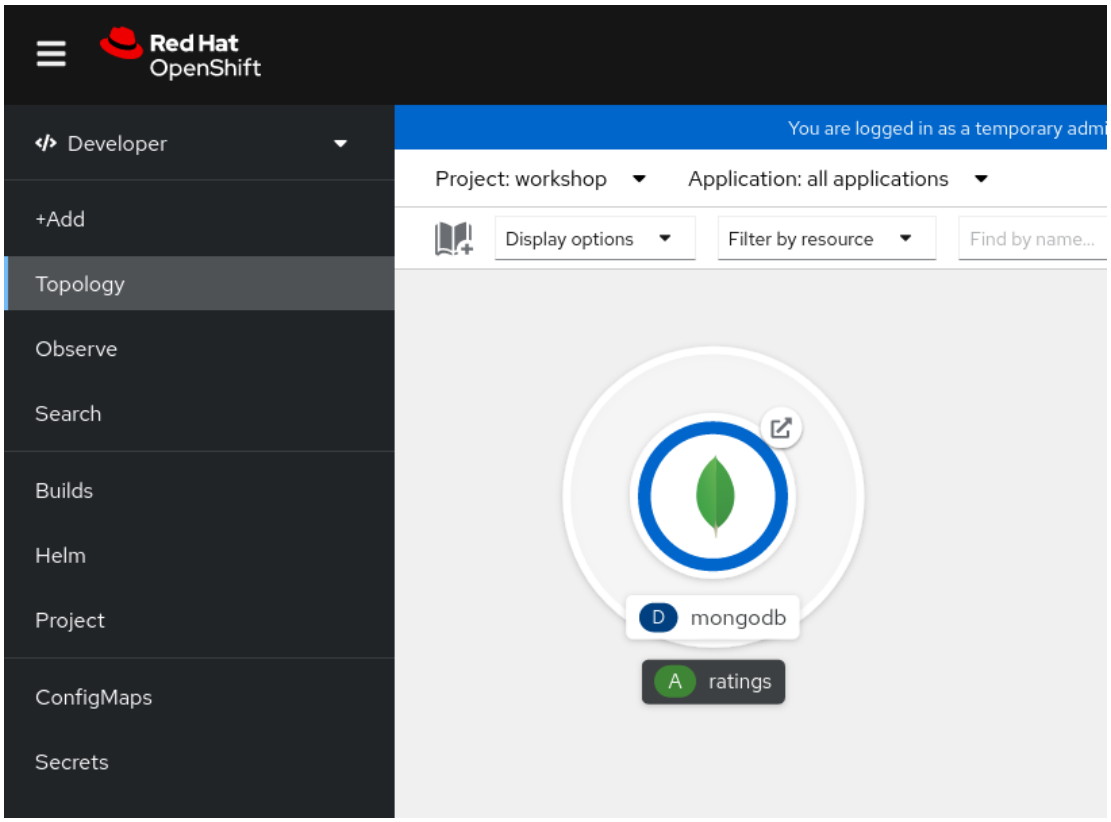


그림 7.9: 실행 중인 MongoDB 인스턴스

`oc get all` 커맨드를 실행하여 새로운 애플리케이션의 상태를 보고, MongoDB 템플릿 배포가 완료되었는지 확인합니다. `oc get all`의 예시 출력은 다음과 같습니다.

```
user@host: oc get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/mongo-6c6fcb45b8-8wpdm      1/1     Running   0          29s

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/mongo     ClusterIP   172.30.88.119   <none>        27017/TCP   30s

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongo     1/1     1            1           30s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/mongo-6c6fcb45b8     1         1         1       30s

NAME                                        IMAGE REPOSITORY
TAGS      UPDATED
imagestream.image.openshift.io/mongo    image-registry.openshift-image-registry.svc:5000/workshop/mongo
latest    30 seconds ago
```

모든 것이 올바르게 작동하고 있으면 STATUS 열에 컨테이너가 실행 중(Running)이라고 표시됩니다.

MongoDB 서비스 호스트 이름 검색

배포가 완료되면 클러스터 내부에서 데이터베이스에 액세스할 수 있도록 생성된 서비스를 찾아야 합니다. `svc`는 `services`를 축약한 형태입니다.

```
user@host: oc get svc mongodb
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
mongo     ClusterIP   172.30.88.119   <none>        27017/TCP   77s
```

서비스는 `[서비스 이름].[프로젝트 이름].svc.cluster.local` 형식의 DNS 이름인 `mongodb.workshop.svc.cluster.local`에서 액세스할 수 있습니다. 이것은 클러스터 내에서만 해결됩니다.

평가 API 배포

이제 두 번째 애플리케이션인 `rating-api`를 배포할 때가 되었습니다. 즉 MongoDB 인스턴스에 연결하여 항목을 검색하고 평가하는 Node.js 애플리케이션입니다. 다음은 이 애플리케이션을 배포하는 데 필요한 몇 가지 세부 사항입니다.

- GitHub의 rating-api
- 컨테이너가 포트 3000을 노출함
- MongoDB 연결을 MONGODB_URI라는 환경 변수를 사용하여 구성

다음에 나오는 몇 개의 섹션에서 참조해야 하므로 이 세부 사항을 적어 두시기 바랍니다.

애플리케이션을 자체 GitHub 리포지토리로 포킹

CI/CD 웹후크 추가와 같은 변경을 수행하려면 `ratings-api` 코드의 자체 사본이 필요합니다. Git에서는 이를 "포크"라고 합니다. 애플리케이션을 개인 GitHub 리포지토리로 포킹할 수 있습니다. 이전 GitHub 리포지토리로 이동하여 **Fork** 버튼을 클릭합니다.

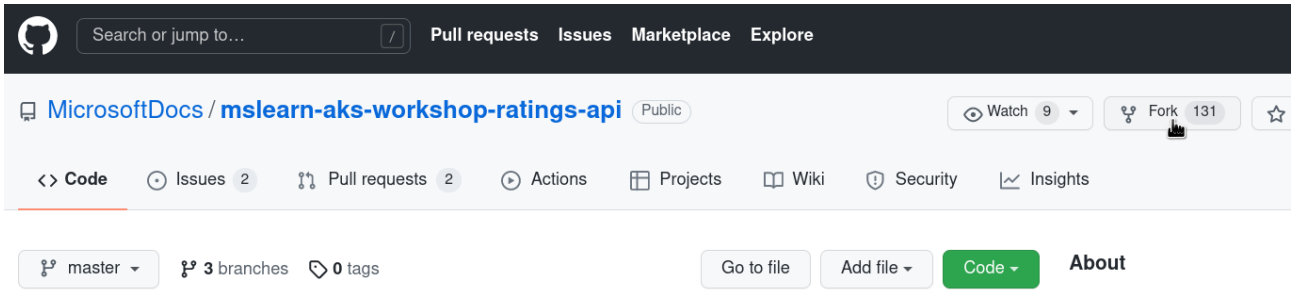


그림 7.10: 애플리케이션을 GitHub 리포지토리로 포킹하기

아래 나오는 지침에서 이 주소를 사용해야 하므로 새로운 리포지토리 주소를 적어 두세요.

OpenShiftCLI를 사용해 rating-api 배포

OpenShift는 콘텐츠를 살펴보고 Java, PHP, Perl, Python 또는 이와 유사한 콘텐츠에 근거하여 "빌더 이미지"를 선택함으로써 Git 리포지토리에서 직접 코드를 배포할 수 있습니다. 이 경우 `rating-api`는 JavaScript 애플리케이션입니다. 빌더 이미지는 npm을 사용해 JavaScript 종속성을 다운로드하고, 그 결과 내부 OpenShift 컨테이너 레지스트리에 새로운 컨테이너 이미지가 저장됩니다. 이 빌드 전략을 **Source 2 Image(S2I)**라고 하며, 이 S2I에 관해서는 용어집에 자세히 기술되어 있습니다.

다음과 같이 oc new-app을 사용해 새로운 S2I 빌드를 시작할 수 있습니다.

```
user@host: oc new-app https://github.com/<your GitHub username>/mslearn-aks-workshop-ratings-api
--strategy=source --name=rating-api

--> Found image 0aea15f (3 weeks old) in image stream “openshift/nodejs” under tag “14-ubi8” for “nodejs”

    Node.js 14
    ----------
    Node.js 14 available as container is a base platform for building and running various Node.
js 14 applications and frameworks. Node.js is a platform built on Chrome’s JavaScript runtime
for easily building fast, scalable network applications. Node.js uses an event-driven, non-
blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time
applications that run across distributed devices.

    Tags: builder, nodejs, nodejs14

    * The source repository appears to match: nodejs
    * A source build using source code from https://github.com/MicrosoftDocs/mslearn-aks-
workshop-ratings-api will be created
      * The resulting image will be pushed to image stream tag “rating-api:latest”
      * Use ‘oc start-build’ to trigger a new build
```

웹 콘솔에서 **토폴로지** 보기로 전환하면 애플리케이션 빌드가 시작되어 몇 분 후 배포가 완료되는 것을 볼 수 있습니다.

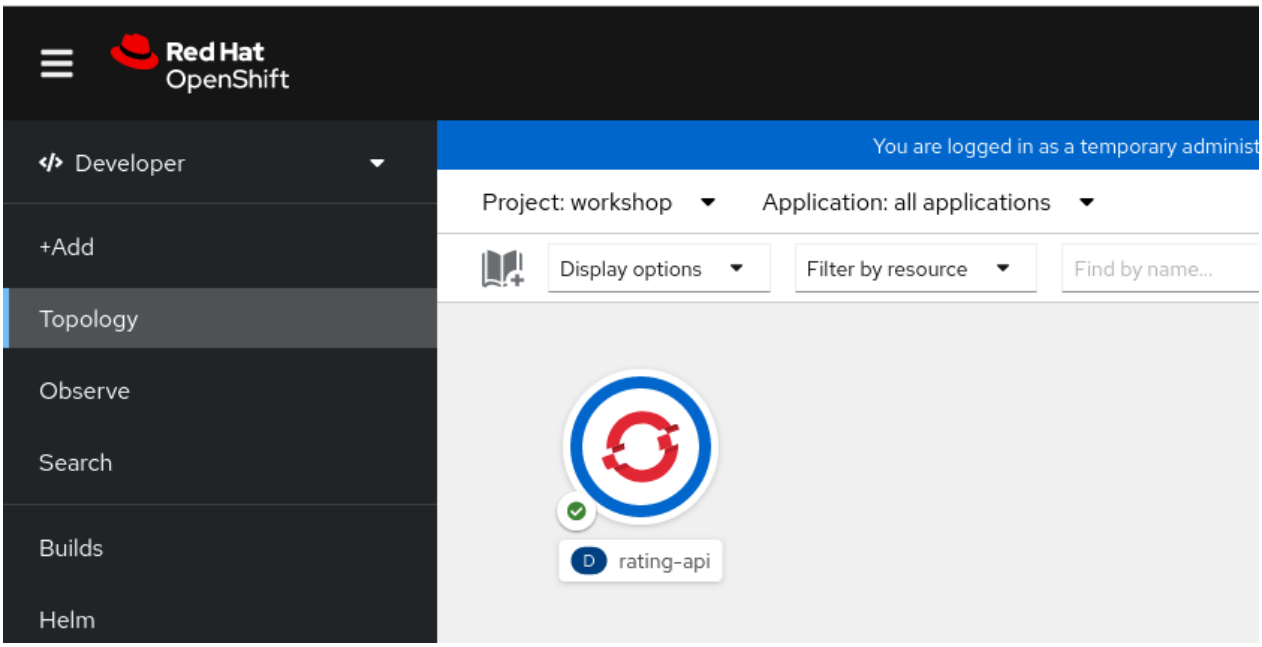


그림 7.11: 토폴로지 보기

포드 빌드 및 시작은 1~2분밖에 안 걸립니다.

필수 환경 변수 구성

이 시점에는 평가 API뿐 아니라 데이터베이스도 배포됩니다. 하지만 데이터베이스에 연결하는 방법을 평가 API에 알려주어야 합니다. 컨테이너 기반 애플리케이션 구성은 일반적으로 환경 변수를 사용해 이루어집니다.

배포를 클릭하여 편집합니다. 다음과 같은 환경 변수를 생성합니다.

```
Name          Value
MONGODB_URI   mongodb://mongodb.workshop.svc.cluster.local:27017/ratingsdb
```

이 새로운 환경 변수를 저장하면 이 새로운 환경 변수를 사용할 `ratings-api` 서비스의 재배포가 트리거됩니다.

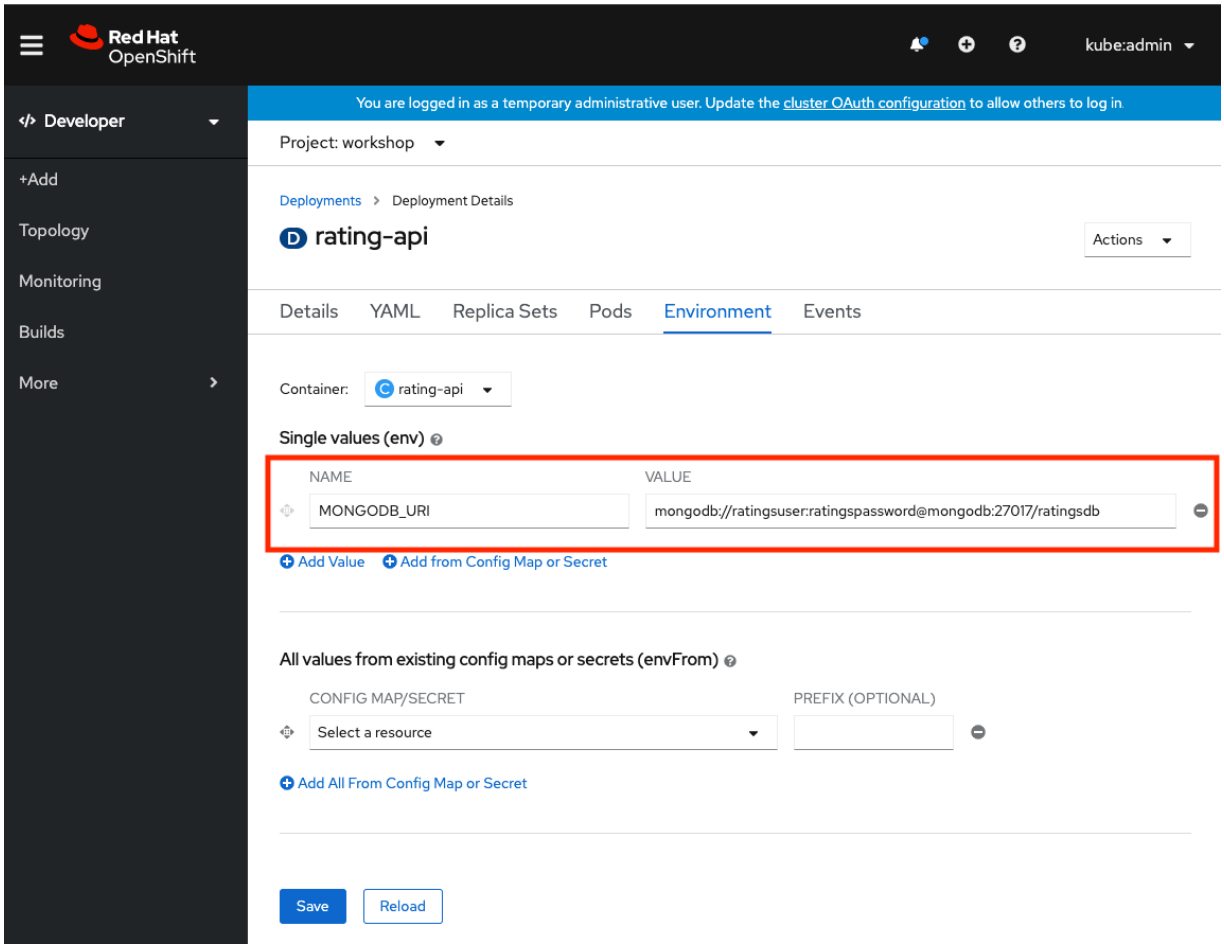


그림 7.12: 웹 콘솔을 통해 MONGODB_URI 환경 변수 설정하기

커맨드라인에서 다음과 같이 설정할 수도 있습니다.

```
oc set env deploy/rating-api MONGODB_URI=mongodb://mongodb.workshop.svc.cluster.local:27017/ratingsdb
```

어떤 방법을 사용하든 OpenShift는 컨테이너를 다시 시작해야 새로운 환경 변수를 사용할 수 있습니다.

서비스가 실행되고 있는지 확인

`rating-api` 배포의 로그로 이동하면 코드가 MongoDB에 연결할 수 있음을 확인하는 로그 메시지가 표시되어야 합니다. 이렇게 하려면 배포의 세부 정보 화면에서 **포드** 탭을 클릭한 후 포드 중 하나를 클릭합니다.

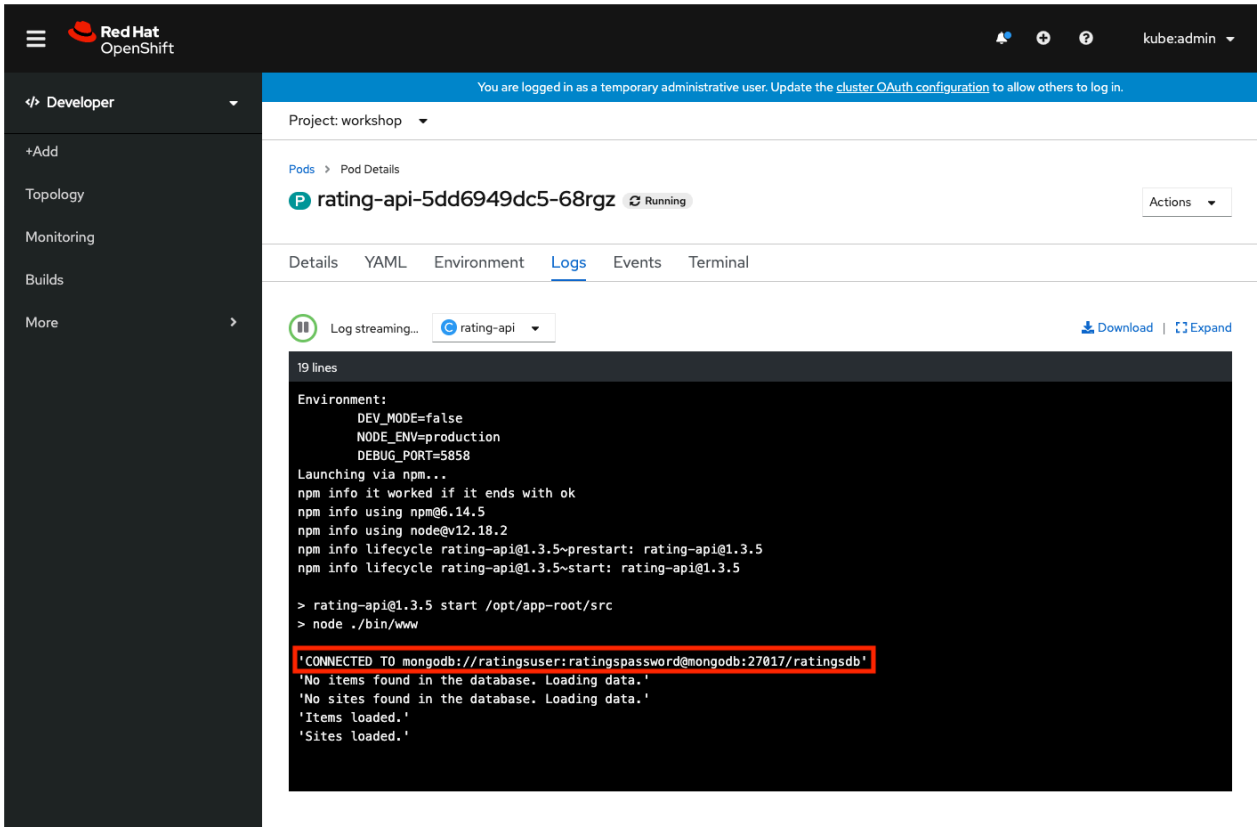


그림 7.13: 코드가 MongoDB에 연결할 수 있음을 확인하는 로그 메시지

rating-api서비스 포트 조정

OpenShift는 포트 `8080`을 사용해 서비스를 생성합니다. 하지만 라이브러리 업데이트로 인해 이 서비스는 포트 `3000`에서 실행됩니다. 기본 서비스를 편집해야 합니다.

네트워킹 → **서비스** 메뉴로 이동한 후 서비스 목록에서 `rating-api` 서비스를 다음과 같이 편집합니다(`8080`을 `3000`으로 교체하기만 하면 됨).

```
  ports:
    - name: 3000-tcp
      protocol: TCP
      port: 3000
      targetPort: 3000
```

새 포트를 사용하기 위해 다음과 같이 서비스를 다시 시작합니다.

```
user@host: oc rollout restart deploy/rating-api
```

이제 서비스는 `8080` 대신에 포트 `3000`에 올바르게 바인딩됩니다.

rating-api 서비스 호스트 이름 검색

다음과 같이 `rating-api` 서비스가 있는지 검증해야 합니다. 다음 섹션에서 `rating-web` 애플리케이션이 배포될 때 이 서비스가 사용되기 때문입니다.

```
oc get service rating-api
```

이 서비스는 포트 `3000`을 통해 `[서비스 이름].[프로젝트 이름].svc.cluster.local` 형식의 `rating-api.workshop.svc.cluster.local:3000` DNS 이름에서 액세스할 수 있습니다. 이것은 클러스터 내에서만 해결됩니다.

등급 프론트엔드 배포

`rating-web`은 `rating-api`에 연결되는 Node.js 애플리케이션입니다. 다음은 이 애플리케이션을 배포하는 데 필요한 몇 가지 세부 사항입니다.

- GitHub의 rating-web
- 컨테이너가 포트 8080을 노출함
- 웹 애플리케이션은 API라는 환경 변수를 통해 프록시를 사용하여 내부 사용자 DNS를 거쳐 API에 연결합니다.

OpenShiftCLI를 사용해 rating-web 배포

`rating-api` 애플리케이션과 마찬가지로 이 애플리케이션은 `oc new-app`을 사용해 S2I로 배포할 수 있습니다. 하지만 여기에서는 Git 리포지토리에 이미 있는 Dockerfile을 사용하는 Dockerfile 전략으로 생성됩니다. 따라서 `--strategy` 인수를 지정하면 안 됩니다.

```
user@host: oc new-app https://github.com/<your GitHub username>/mslearn-aks-workshop-ratings-web
--name rating-web

--> Found container image e1495e4 (2 years old) from Docker Hub for “node:13.5-alpine”

    * An image stream tag will be created as “node:13.5-alpine” that will track the source image
    * A Docker build using source code from https://github.com/MicrosoftDocs/mslearn-aks-
workshop-ratings-web will be created
      * The resulting image will be pushed to image stream tag “rating-web:latest”
      * Every time “node:13.5-alpine” changes a new build will be triggered

--> Creating resources ...
    imagestream.image.openshift.io “node” created
    imagestream.image.openshift.io “rating-web” created
    buildconfig.build.openshift.io “rating-web” created
    deployment.apps “rating-web” created
    service “rating-web” created
--> Success
```

종속성을 컨테이너로 빌드하는 데 얼마 걸리지 않습니다. 빌드가 완료될 때까지 2~3분 가량 기다렸다가 **토폴로지** 보기로 되돌아가면 서비스가 온라인 상태임을 확인할 수 있습니다.

필수 환경 변수 구성

`rating-web` 배포 구성을 위해 `API` 환경 변수를 생성합니다. 이 변수의 값은 `rating-api` 서비스의 호스트 이름/포트가 됩니다.

Azure Red Hat OpenShift 웹 콘솔을 통해 환경 변수를 설정하는 대신에 다음과 같이 OpenShift CLI를 통해 설정할 수 있습니다.

```
oc set env deploy rating-web API=http://rating-api:3000
```

경로를 사용해 rating-web 서비스 노출

서비스 노출이란 사용자가 서비스에 액세스하기 위해 사용할 수 있는 공개적으로 액세스 가능한 URL이 있음을 뜻합니다. 서비스가 노출되지 않은 경우 클러스터 내에서만 액세스할 수 있습니다.

```
user@host: oc expose svc/rating-web
route.route.openshift.io/rating-web exposed
```

끝으로 방금 노출한 서비스의 URL을 다음과 같이 가져옵니다.

```
user@host: oc get route rating-web
NAME          HOST/PORT                                                PATH    SERVICES      PORT
TERMINATION    WILDCARD
rating-web    rating-web-workshop.apps.zytjwj9a.westeurope.aroapp.io          rating-web    8080-tcp
None
```

정규화된 도메인 이름(FQDN)은 기본적으로 애플리케이션 이름과 프로젝트 이름으로 구성됩니다. FQDN의 나머지 부분인 하위 도메인은 Azure Red Hat OpenShift 클러스터별 애플리케이션 하위 도메인입니다.

서비스 사용해 보기

브라우저에서 호스트 이름을 엽니다. 그러면 평가 애플리케이션 페이지가 표시될 것입니다. 두루 살펴보고 몇 가지 평가를 제출한 후 순위표를 확인하세요.

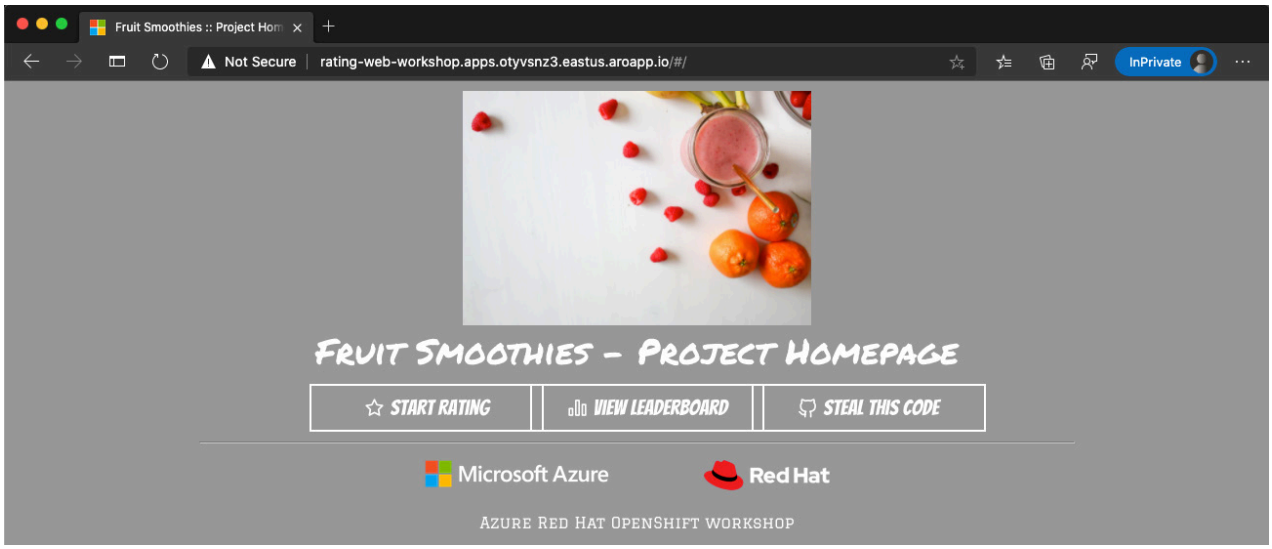


그림 7.14: 평가 애플리케이션 서비스 사용해 보기

GitHub 웹후크 설정

코드를 GitHub 리포지토리로 푸시할 때 S2I 빌드를 트리거하려면 다음과 같이 GitHub 웹후크를 설정해야 합니다.

1. GitHub 웹후크 트리거 암호를 검색합니다. 다음과 같이 GitHub 웹후크 URL에서 이 암호를 사용해야 합니다.

```
user@host: oc get bc/rating-web -o=jsonpath='{.spec.triggers..github.secret}'
3ffcc8d5-a243
```

   몇 단계 후 필요하므로 암호 키를 적어 두세요.

2. 다음과 같이 빌드 구성에서 GitHub 웹후크 트리거 URL을 검색합니다.

```
user@host: oc describe bc/rating-web
...
Webhook GitHub:
        URL:    https://api.quwhfg7o.westeurope.aroapp.io:6443/apis/build.openshift.io/v1/
namespaces/workshop/buildconfigs/rating-web/webhooks/<secret>/github
...
```

3. <secret> 자리 표시자를 첫 단계에서 검색한 암호로 교체합니다. 이 경우 암호는 3ffcc8d5-a243이므로 다음과 같은 URL을 얻게 됩니다.

```
https://api.quwhfg7o.westeurope.aroapp.io:6443/apis/build.openshift.io/v1/namespaces/workshop/
buildconfigs/rating-web/webhooks/3ffcc8d5-a243/github
```

   이 URL을 사용해 GitHub 리포지토리에서 웹후크를 설정하게 됩니다.

4. GitHub 리포지토리로 이동합니다. **설정** → **웹후크**에서 **웹후크 추가**를 선택합니다.
5. **페이로드 URL** 필드에서 암호를 사용할 수 있도록 <secret> 값이 변경된 GitHub URL을 붙여넣습니다.
6. **콘텐츠 유형** 필드에서 기본값인 application/x-www-form-urlencoded를 application/json으로 변경합니다.

7. **웹후크 추가**를 클릭합니다.

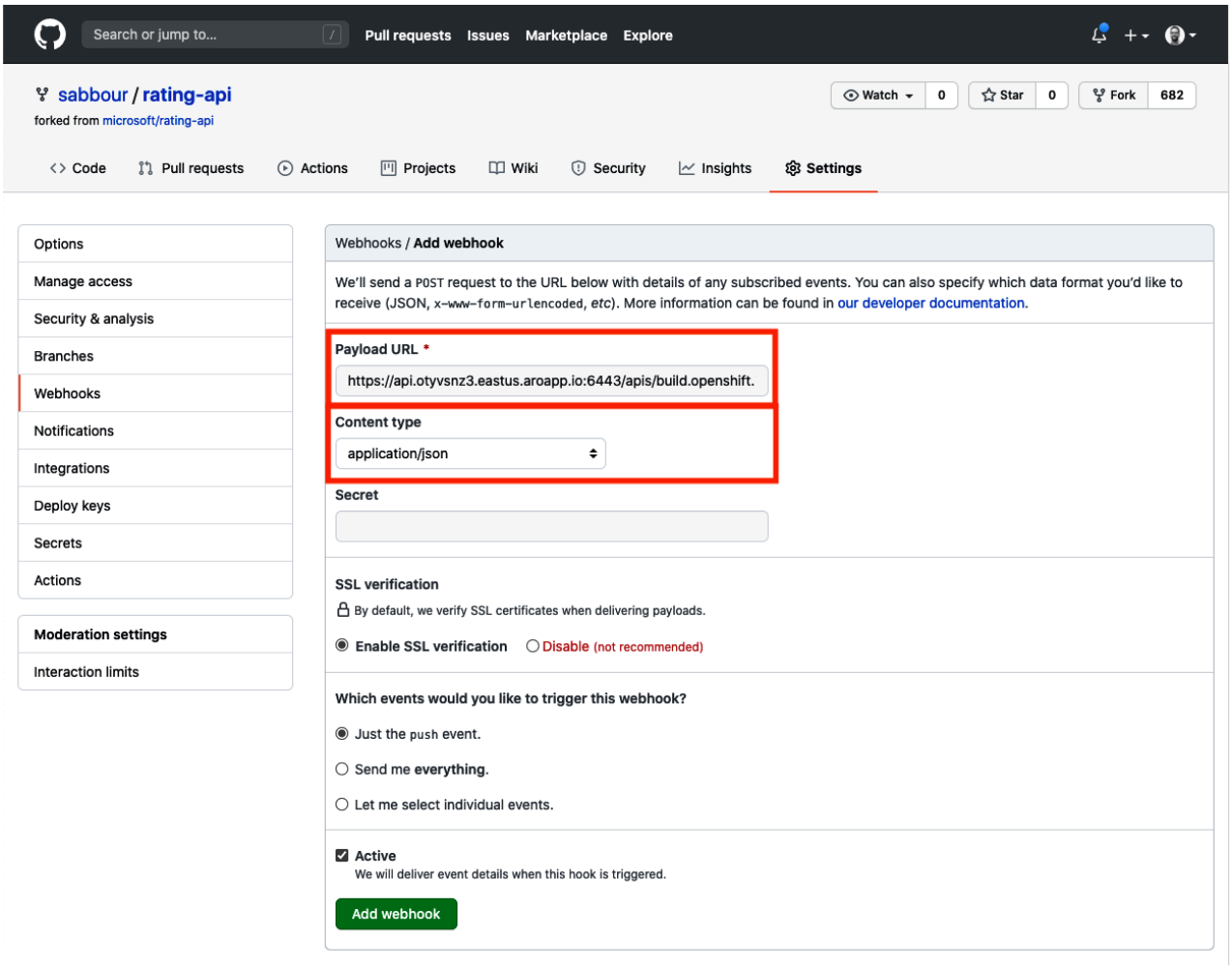


그림 7.15: 웹후크 추가하기

웹후크 구성이 완료되었다는 내용의 GitHub 메시지가 표시됩니다.

이제 GitHub 리포지토리로 변경 사항을 푸시할 때마다 새로운 빌드가 자동으로 시작되며, 빌드가 완료되면 새로운 배포가 시작됩니다.

웹사이트 애플리케이션을 변경하고 롤링 업데이트 보기

GitHub의 리포지토리에 있는 `https://github.com/<GitHub 사용자 이름>/rating-web/blob/master/src/App.vue` 파일로 이동합니다.

파일을 편집합니다. 즉 `background-color: #999;` 행을 `background-color: #0071c5`로 변경합니다.

파일 변경 사항을 `master` 브랜치로 커밋합니다.

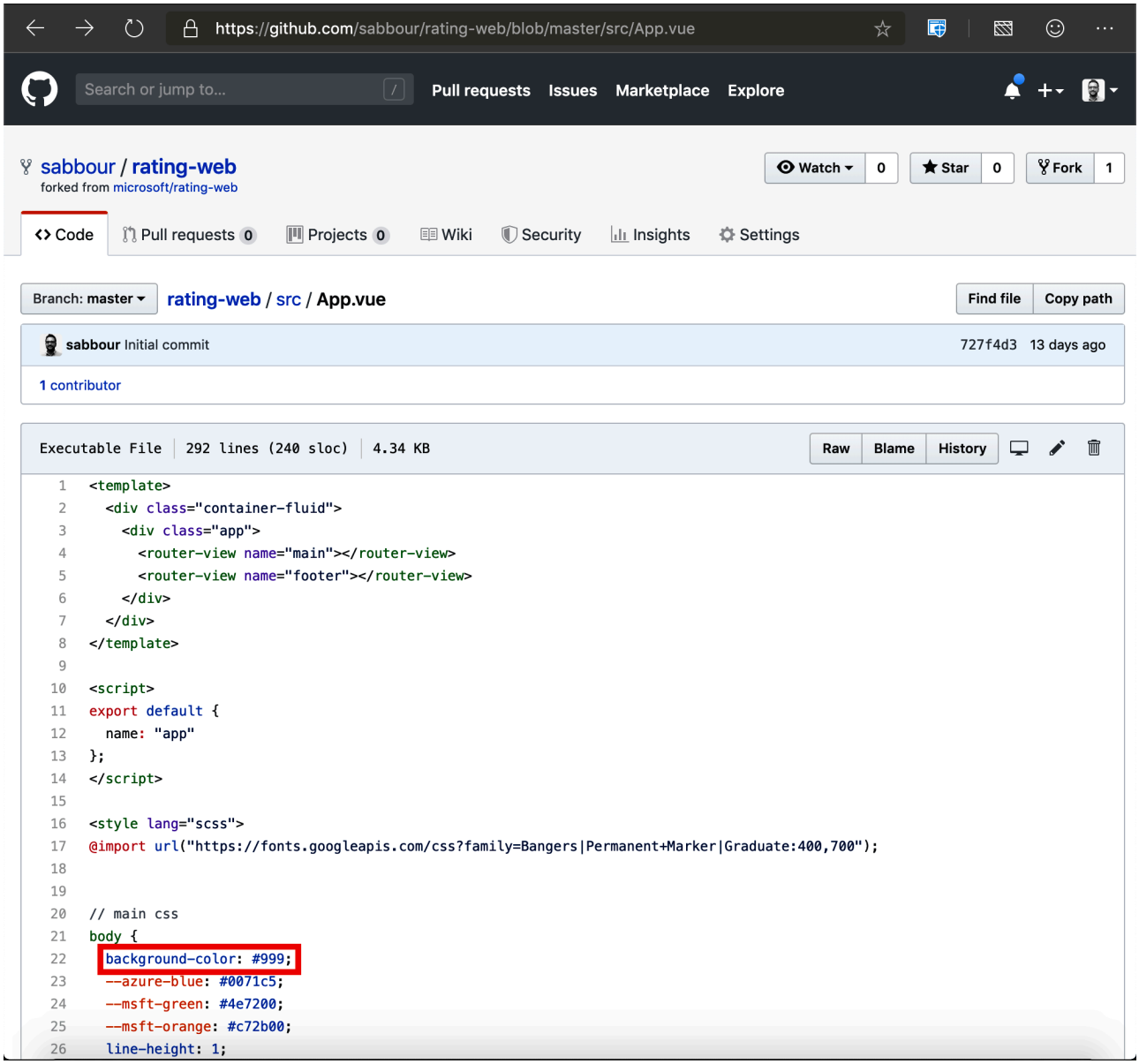


그림 7.16: 파일 변경 사항을 master 브랜치로 커밋하기

즉시 OpenShift 웹 콘솔의 **빌드** 탭으로 이동합니다. 푸시로 트리거된 새로운 빌드가 대기열에 추가되는 것을 볼 수 있습니다. 이 단계가 완료되면 새로운 배포가 트리거되어 웹사이트의 색상이 업데이트된 것을 볼 수 있습니다.

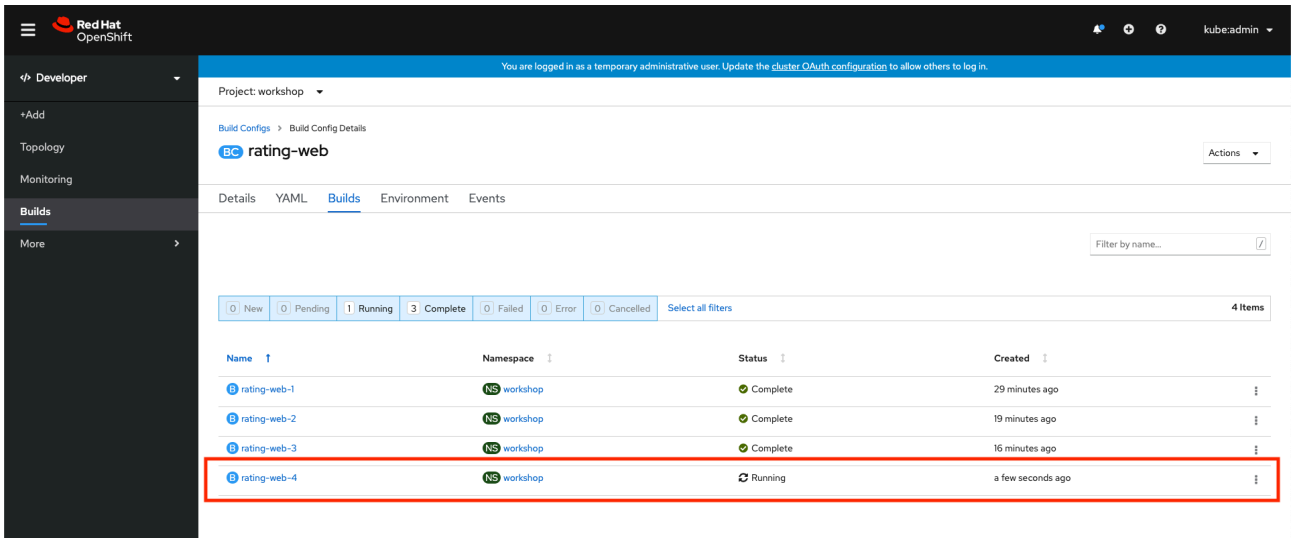


그림 7.17: 새로운 빌드가 실행 중임을 표시하는 빌드 탭

이제 `ratings-web` 페이지로 돌아갑니다. 모든 것이 제대로 완료되었다면 배경색이 변경되었음을 보게 됩니다!

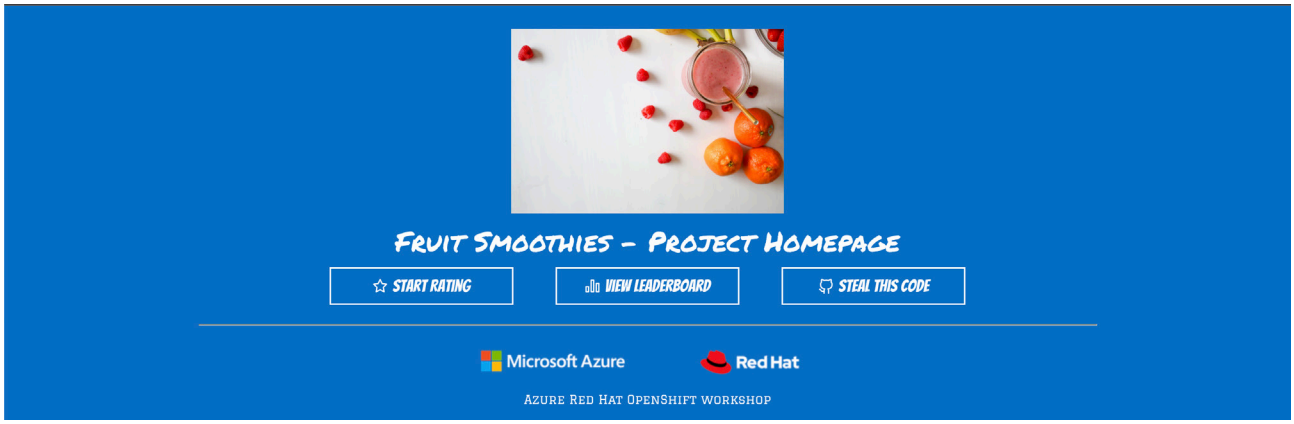


그림 7.18: 새로운 색상으로 변경된 과일 스무디 홈페이지

요약

이 장에서 우리는 더 적은 규모의 세 가지 마이크로서비스 애플리케이션(MongoDB 데이터베이스, `ratings-api`, `ratings-web`코드)으로 구성된 기본 애플리케이션을 배포했습니다. 프로덕션 애플리케이션과 그다지 비슷하지 않지만 Azure Red Hat OpenShift에 애플리케이션을 배포할 때 개념에 대해 신속하게 상기시키는 역할을 합니다. 이 지침은 애플리케이션 배포 기준으로 사용할 수 있습니다.

또한 Red Hat OpenShift는 OperatorHub뿐 아니라 Helm Charts, 외부 CI/CD 시스템(예: Azure DevOps)에서도 애플리케이션을 배포할 수 있도록 지원합니다. 정확히 어떤 배포 전략이 조직에 가장 적합한지는 내부에서 사용하는 툴과 기술에 달려 있습니다.

다음 장에서는 쿠버네티스 위에 구축된 Azure Red Hat OpenShift의 몇 가지 부가적 가치(애플리케이션 플랫폼인 OpenShift를 차별화하는 가치)에 대해 알아보겠습니다. 또한 엔터프라이즈 애플리케이션의 복잡한 요구 사항을 지원하도록 설계된 특징과 기능에 대해 자세히 알아봅니다.

8장

애플리케이션 플랫폼 살펴보기

이전 장에서는 Red Hat OpenShift가 쿠버네티스를 기반으로 구축된 다양한 서비스를 제공하는 방식에 대해 알아보았습니다. 이러한 각 서비스는 서로 결합되어 진정한 애플리케이션 플랫폼을 구성하며 플랫폼 서비스, 애플리케이션 서비스, 데이터 서비스, 개발자 서비스, 쿠버네티스 클러스터 서비스의 다섯 가지 범주 중 하나로 그룹화할 수 있습니다.

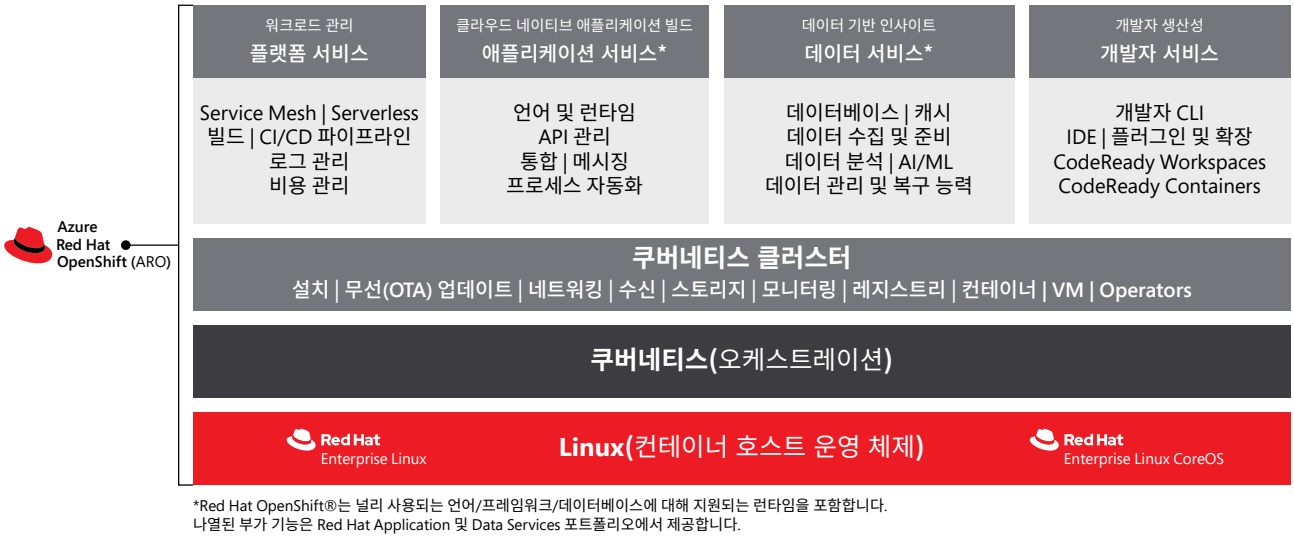


그림 8.1: Azure Red Hat OpenShift에 포함된 서비스

OpenShift Platform Plus(멀티클러스터 관리, 클러스터 보안, 전역 레지스트리)에 그룹화된 구성 요소는 서브스크립션이 필요한 부가적인 제품입니다. 이 제품들은 Azure Red Hat OpenShift와 호환되지 않지만 Azure Red Hat OpenShift 제공 사항에 포함되어 있지 않습니다.

다음 섹션에서는 이러한 서비스가 제공하는 주요 이점 몇 가지를 살펴보고 자세한 정보를 볼 수 있는 링크를 제공해 드립니다.

클러스터 서비스 – 통합된 컨테이너 레지스트리

Red Hat OpenShift는 클러스터가 배포되자마자 통합된 내부 컨테이너 레지스트리와 함께 제공됩니다. 이 레지스트리는 클러스터 오퍼레이터와 같은 클러스터 내부 서비스뿐 아니라 기본적으로 고객의 애플리케이션 컨테이너에도 사용됩니다. 이 레지스트리는 표준이며 추가 설정이 필요 없을 뿐 아니라 인프라 운영자도 유지 관리할 수 있습니다.

쿠버네티스 컨테이너 서비스를 배포하는 모든 고객은 자체 컨테이너 레지스트리를 배포할 뿐 아니라 자체 컨테이너 이미지를 프라이빗으로 유지해야 할 수 있습니다. OpenShift를 사용하면 빌트인 컨테이너 레지스트리가 클러스터 내에서 이미 제공되므로 이러한 부가적인 2일 차 설치 및 설정이 필요 없습니다. 이것은 단순하지만 시간이 절약되는 OpenShift 내부 기능의 예입니다.

통합된 OpenShift 컨테이너 플랫폼 레지스트리 개요

흔히 클러스터 관리자는 OpenShift 외부에 있는 사용자가 컨테이너 이미지를 레지스트리로 푸시할 수 있도록 클러스터 외부로 이 컨테이너 레지스트리를 노출하는 방식을 선택할 수 있습니다. 이러한 방식은 Azure Red Hat OpenShift 내에서 완전히 지원되며, 이렇게 하는 방법에 관한 도큐멘테이션은 레지스트리 노출에 관한 표준 OpenShift 도큐멘테이션에서 찾을 수 있습니다.

플랫폼 서비스 – OpenShift Pipelines

Red Hat OpenShift의 고객은 다양한 방식으로 애플리케이션을 빌드할 수 있으며, 널리 사용되는 다수의 CI/CD 툴(예: Jenkins, CircleCI, GitHub Actions)에는 OpenShift를 지원하는 플러그인이 있습니다. 하지만 OpenShift는 플랫폼에서 OpenShift Pipelines 오퍼레이터를 통해 클라우드 네이티브 컨테이너 파이프라인도 사용해 애플리케이션을 빌드할 수 있는 기능도 제공합니다.

OpenShift Pipelines는 Tekton이라는 커뮤니티 프로젝트에 기반을 두고 있습니다. 파이프라인의 각 단계(예: Git 리포지토리에서 코드 풀링, Java 컴파일러 실행 또는 RPM 패키지 조합)는 컨테이너 내에서 실행됩니다. 이는 개발자와 운영자가 컨테이너가 제공하는 이점을 최대한 활용해 복잡하고 정교한 파이프라인을 구성하여 소프트웨어를 빌드할 수 있음을 뜻합니다.

OpenShift Pipelines 설치는 OperatorHub를 통해 가능합니다. **OperatorHub**로 이동해 설치를 시작할 오퍼레이터를 선택하기만 하면 됩니다. 구성이 필요 없으며, 오퍼레이터 설치는 일반적으로 1분 내에 완료됩니다.

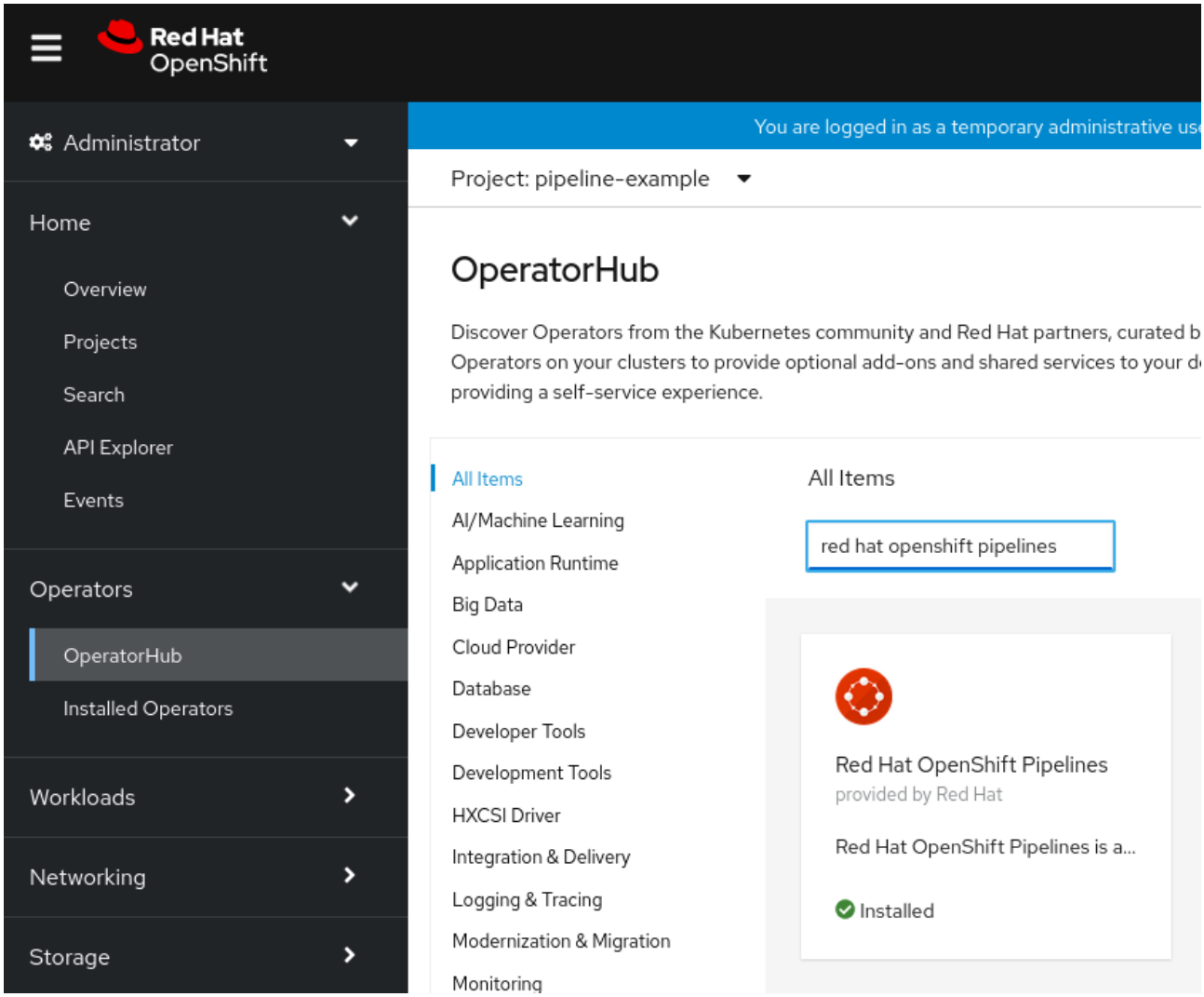


그림 8.2: OperatorHub를 통해 OpenShift Pipelines 설치하기

OpenShift Pipelines 오퍼레이터가 설치되면 사이드바의 **파이프라인** 섹션뿐 아니라 Node.js 예시를 빌드할 때와 같이 카탈로그 항목에 파이프라인을 추가할 수 있는 옵션도 표시됩니다.

그림 8.3: 파이프라인 추가

OpenShift Pipelines를 사용하면 시각적인 빌더를 사용해 복잡하고 분기된 파이프라인도 설정하고 빌드할 수 있습니다. 다음은 더 복잡한 파이프라인의 예를 보여주는 스크린샷입니다.

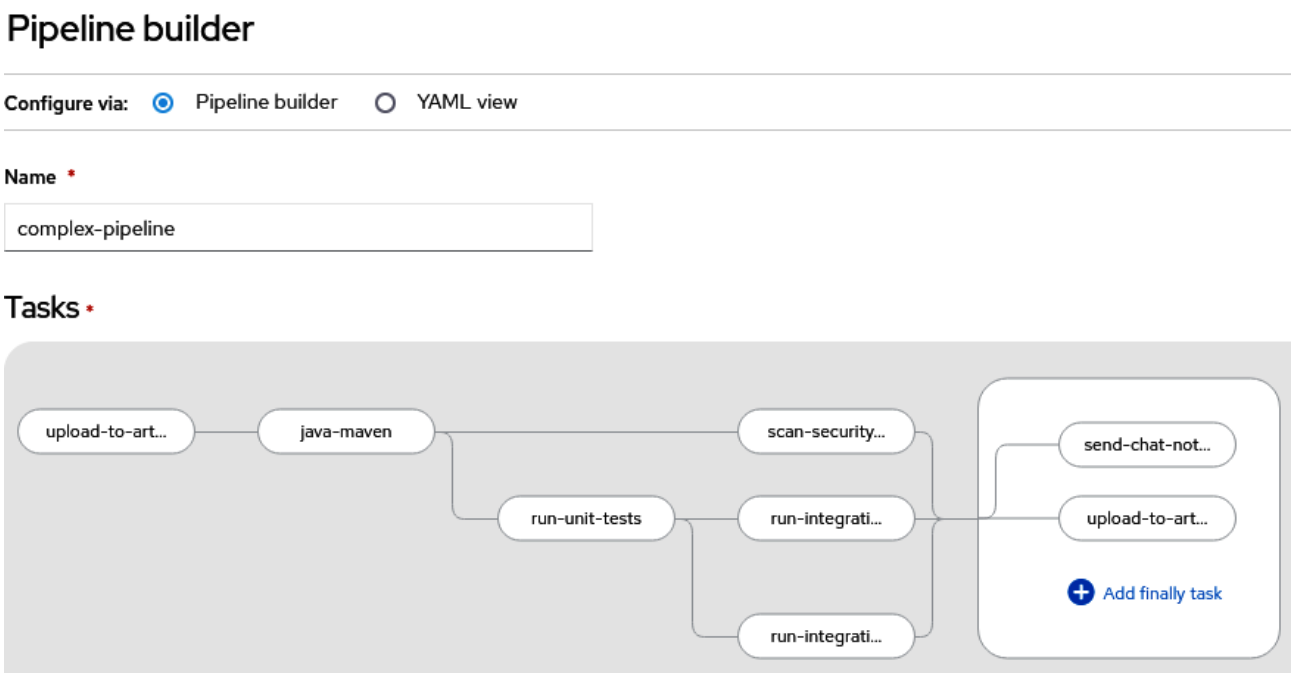


그림 8.4: 복잡한 파이프라인

많은 조직이 다양한 툴과 기술을 사용해 자체 소프트웨어를 빌드하겠지만 OpenShift Pipelines를 통해 컨테이너가 제공하는 모든 이점을 활용해 OpenShift 내에서 빌드를 손쉽게 직접 통합할 수 있습니다. OpenShift Pipelines는 사용 중인 기본 인프라에 관계없이 극도의 최소 설정만 필요한 사용하기 쉽고 일관된 CI/CD 솔루션을 제공합니다.

추가 자료

- OpenShift의 OpenShift Pipelines에 대한 이해
- Tekton 커뮤니티 사이트

플랫폼 서비스 – OpenShift Serverless

컨테이너가 장기 실행 중인 서비스에 유용한 기술일 뿐이라는 것은 흔한 오해입니다. 실제로 다수의 짧게 지속되는 작업과 서버리스 기능은 짧게 지속되는 컨테이너를 기반으로 실행됩니다. 컨테이너가 시작, 일관성, 종료 용이성 측면에서 제공하는 이점으로 인해 이 기능들은 서버리스 워크로드에도 적합합니다. 물론 모든 서버리스 서비스에는 코드를 실행할 기본 서버가 필요하며, 이러한 이유로 서버리스는 때로 "서비스로서의 기능"이라고 불립니다.

Red Hat OpenShift는 OpenShift Serverless 오퍼레이터를 통해 서버리스 또는 서비스로서의 기능 워크로드를 지원합니다. 이 오퍼레이터는 Knative라는 널리 사용되는 오픈소스 프로젝트에 기반을 두고 있습니다.

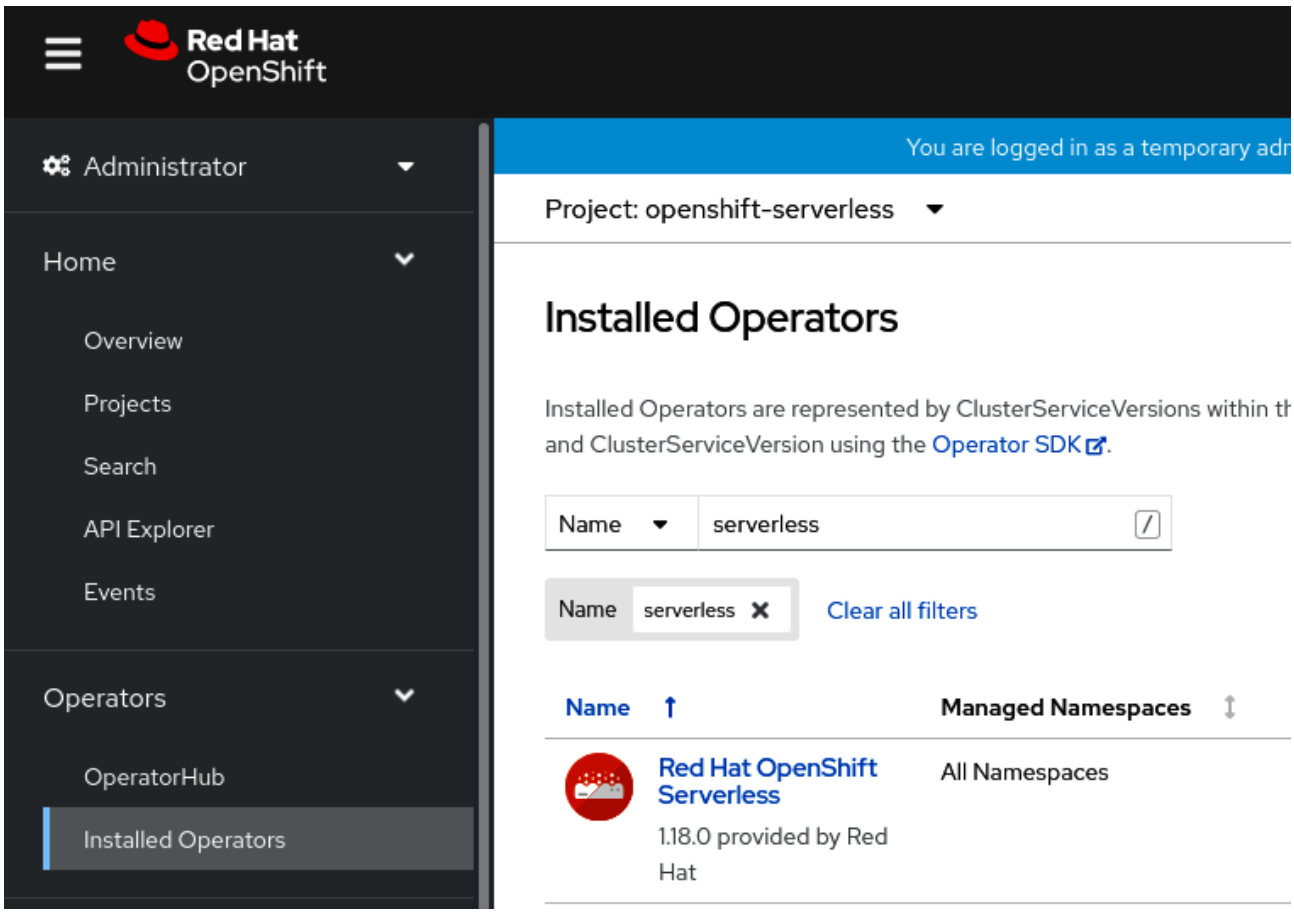


그림 8.5: 설치된 오퍼레이터 보기

클러스터에 배포되고 나면(이 역시 일반적으로 1-2분 소요됨) 오퍼레이터에 대한 약간의 설정 작업이 필요합니다. 특히 **Knative Serving**과 **Knative Eventing** 등의 두 가지 **사용자 정의 리소스 정의(CRD)**에 신경 써야 합니다.

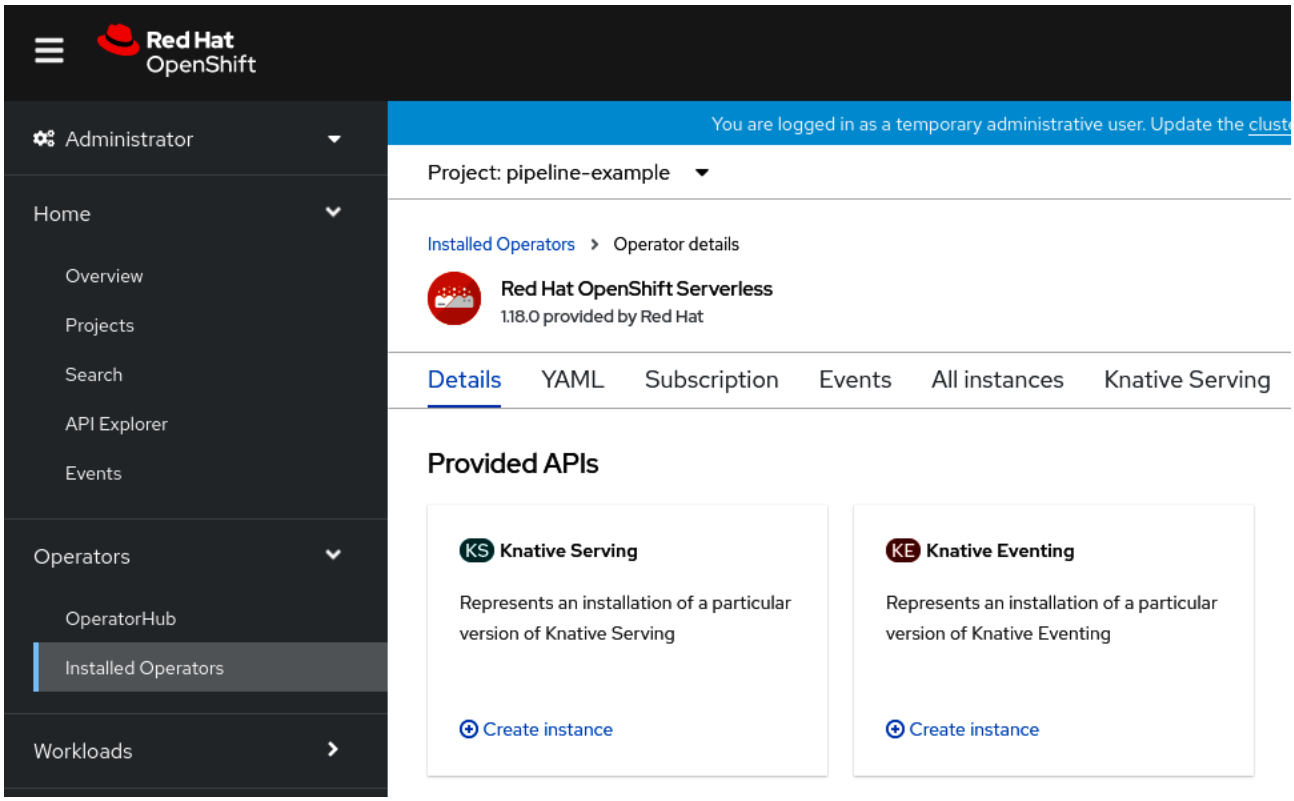


그림 8.6: 오퍼레이터 메뉴의 Knative Serving과 Knative Eventing

- **Knative Serving** 애플리케이션 배포를 간소화하고, 수신 트래픽에 근거하여 동적으로 확장하며, 트래픽 분할로 사용자 정의 출시 전략을 지원합니다. 예를 들어 스토리지 버킷으로 이미지를 업로드하는 기능과 NoSQL 데이터베이스로 업로드되는 로그가 있습니다.
- **Knative Eventing** (빌드 시점이 아닌) 애플리케이션 런타임에 이벤트 소스의 "런타임에 바인딩"을 허용합니다. 예를 들어 스토리지 버킷의 새로운 이미지 업로드에 대응하는 애플리케이션이 있습니다. 이 애플리케이션은 빌드 또는 이벤트 배포 시점에 스토리지 버킷에 관해 알 필요가 없지만 Knative Eventing을 통해 이 이벤트 소스를 런타임에 이 애플리케이션에 손쉽게 "바인딩"할 수 있습니다.

다음 두 섹션에는 이러한 유형의 OpenShift 기반 서버리스 애플리케이션 각각에 대한 예시가 포함되어 있습니다.

플랫폼 서비스 – OpenShift Serverless – Knative Serving 예시

Knative Serving이 애플리케이션을 자동 확장하는 방식, 특히 요청이 없을 때 제로로 축소하는 방식을 데모 형식으로 살펴보겠습니다. 우리가 사용할 수 있는 매우 단순한 한 가지 애플리케이션은 다음과 같이 반려 동물의 ASCII 사진을 제공하는 웹 페이지입니다.

- php-ascii-pets GitHub 리포지토리

Git에서 이 리포지토리를 추가하는 것은 간단합니다. Red Hat OpenShift는 PHP의 호환되는 빌더 이미지를 자동으로 감지합니다.

OpenShift는 이 프로젝트를 빌드하는 방법을 자동으로 감지합니다.

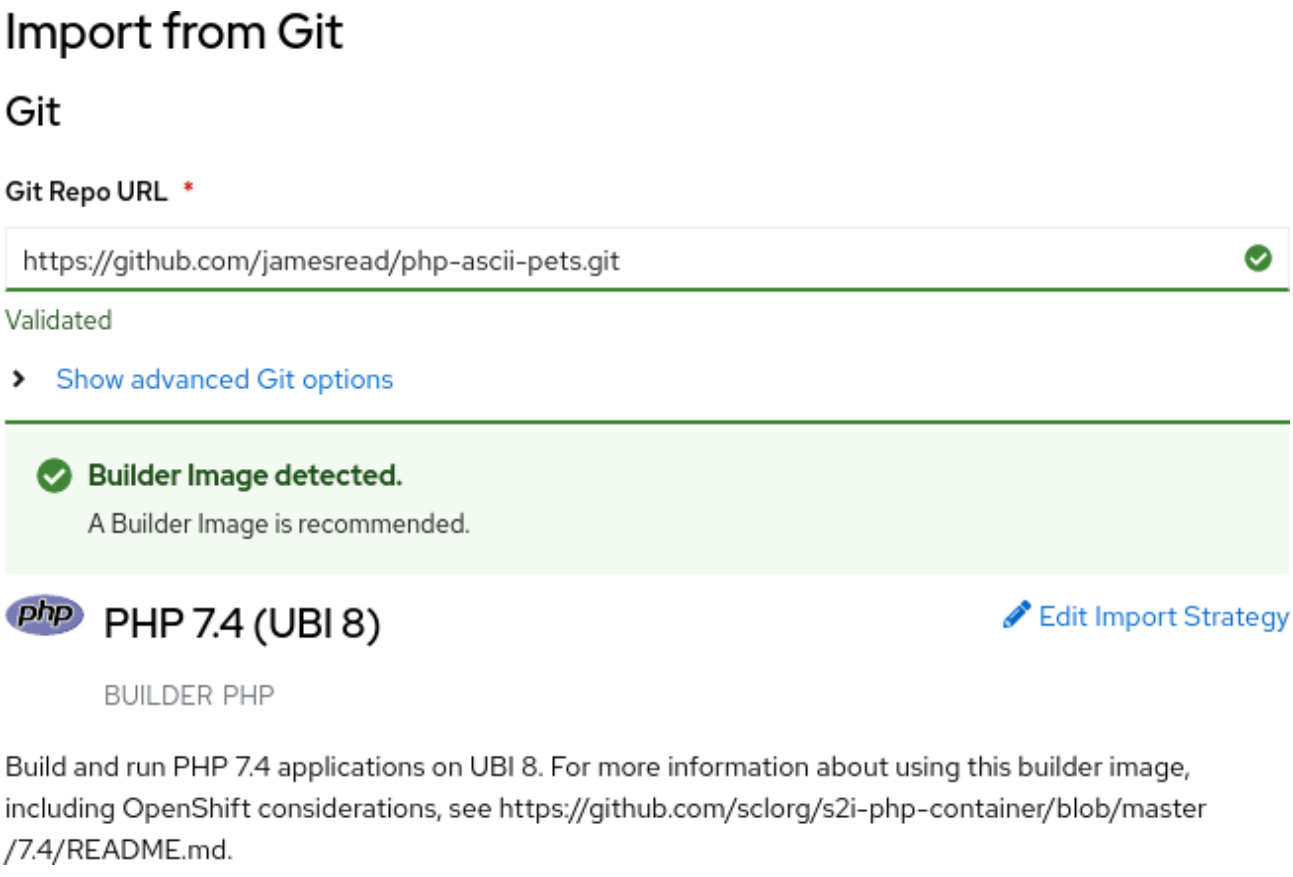


그림 8.7: 자동으로 감지되고 권장되는 빌더 이미지

이것을 "서버리스" 배포로 만드는 중요 부분은 배포 유형이 선택되는 시점입니다. OpenShift Serverless가 설치되면 "서버리스" 배포를 사용할 수 있게 됩니다.

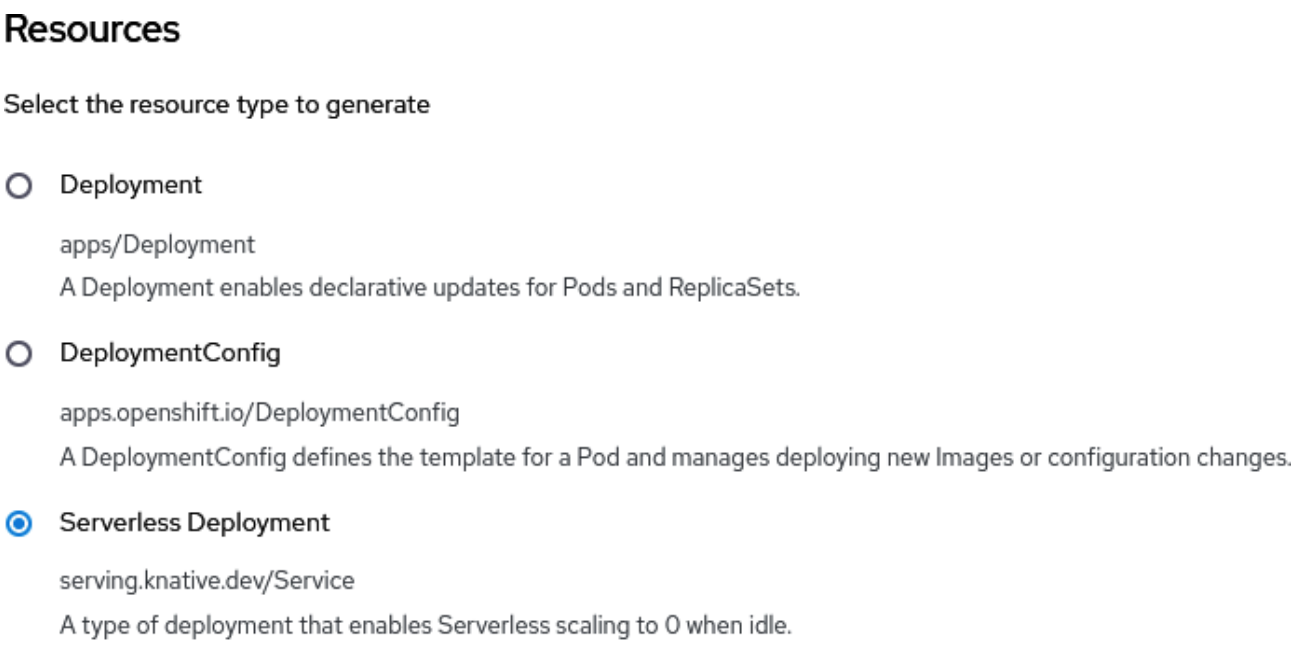


그림 8.8: 서버리스 배포 유형

첫 번째 빌드가 완료될 때까지 몇 분 정도 소요됩니다. 빌드가 완료되고 나면 배포된 포드가 있어야 합니다. 토폴로지 보기는 다음과 같이 표시되어야 합니다.

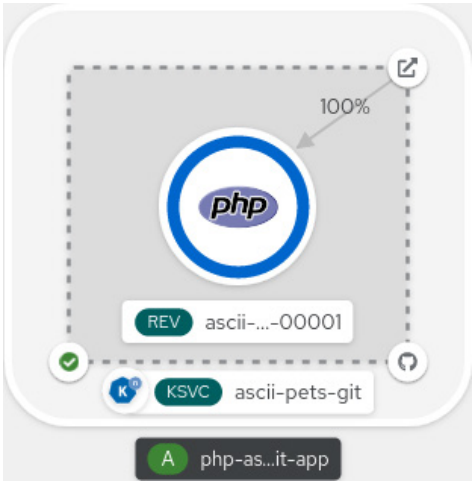


그림 8.9: Knative Serving 애플리케이션

실제 애플리케이션 페이지는 *그림 8.9*와 같습니다. 매우 단순한 애플리케이션이지만 ASCII 아트인 "반려 동물"은 포드 호스트 이름과 결부되어 있습니다. 우리의 단순한 데모 애플리케이션에서 이 애플리케이션에 많은 양의 추가 로드를 투입하면 Knative Serving는 추가 포드를 생성하므로 다양한 "반려 동물"이 제공되고 있음을 시각적으로 확인할 수 있습니다!

하지만 이 애플리케이션에 1분 동안 요청이 없으면 Knative Serving은 이 애플리케이션을 제로로 축소합니다. 토폴로지 보기를 통해 애플리케이션이 실행되고 있지 않음을 알 수 있습니다.

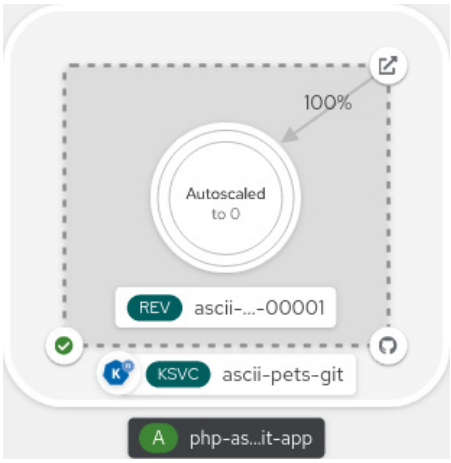


그림 8.10: Knative Serving을 이용한 서버리스 배포(애플리케이션이 제로로 축소됨)

끝으로 누군가 이 페이지를 방문하면 애플리케이션은 OpenShift가 수요 충족에 필요하다고 인식하는 인스턴스의 개수에 따라 최대 1개, 2개, 3개 또는 그 이상의 복제본으로 자동 확장됩니다.

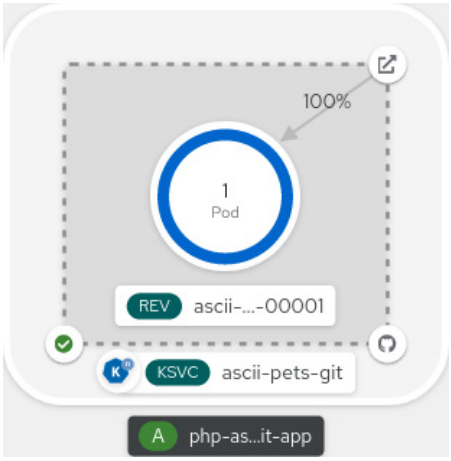


그림 8.11: 트래픽에 대응한 자동 확장

Knative Serving이 포함된 Red Hat OpenShift Serverless를 이용해 조직은 추가 구성이 거의 없이 애플리케이션을 변경하지 않고도 동적으로 확장 및 축소할 수 있습니다. 이 기능은 배포를 적합한 크기로 유지하고 리소스가 불필요하게 사용되어 비용이 결제되는 일을 방지하는 데 엄청난 도움이 됩니다.

추가 자료

- OpenShift Serverless 소개
- OpenShift Serverless에 대한 교육 과정이 포함된 learn.openshift.com
- Knative 커뮤니티 사이트

플랫폼 서비스 – OpenShift Serverless – Knative Eventing 예시

앞 장의 예시 애플리케이션과 기본 사항을 바탕으로 OpenShift Serverless는 인그레스 트래픽 외에 지표에 근거하여 애플리케이션을 확장할 수도 있습니다. 특히 이벤트 기반 아키텍처와 같은 시나리오에서는 애플리케이션의 인스턴스를 확장하여 메시지 대기열에서 이벤트를 처리하거나 타이머에 따라 대기 상태에서 해제되는 것이 바람직합니다.

동일한 배포를 사용해 OpenShift 콘솔에서 다양한 이벤트 소스를 손쉽게 구성할 수 있습니다.

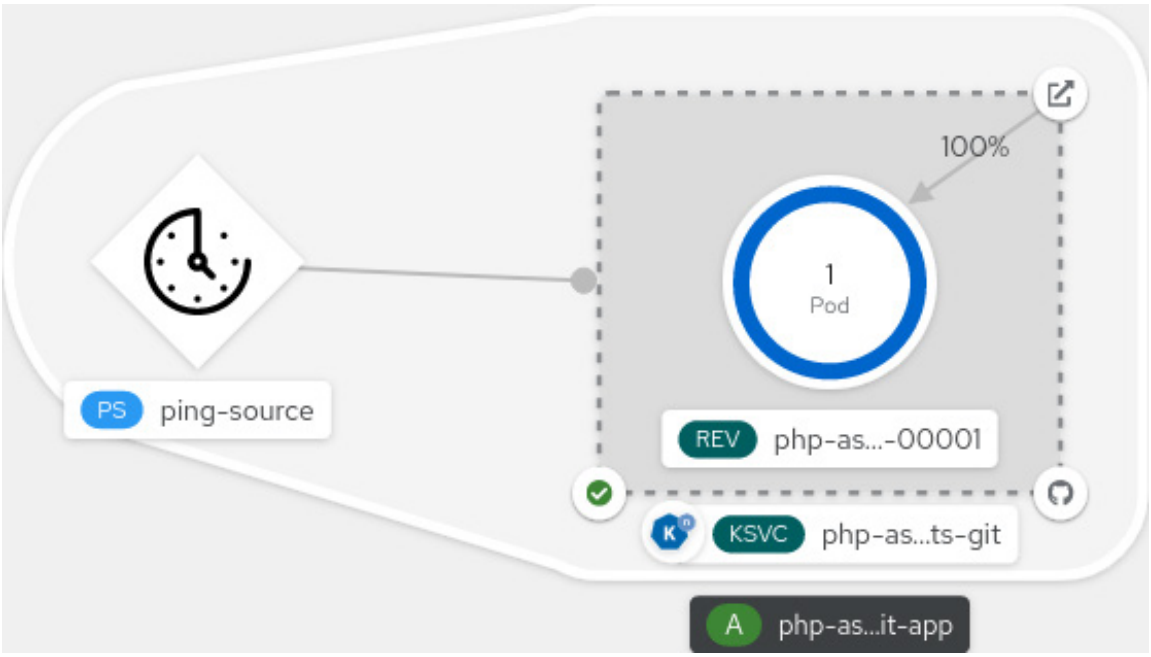


그림 8.12: 타이머에 따라 애플리케이션을 ping하는 "ping" 이벤트 소스

ping 이벤트 소스의 간단한 활용 사례는 매일 밤 자정에 백업 작업 컨테이너를 "대기 상태에서 해제"하는 것입니다. 이 ping 타이머는 주기적으로 실행되는 모니터링 컨테이너에 사용할 수도 있습니다.

추가 자료

- 5분 내에 이해하는 OpenShift Serverless Eventing
- OpenShift Serverless 소개

플랫폼 서비스 – OpenShift Service Mesh

오픈소스 Istio 프로젝트에 기반한 Red Hat OpenShift Service Mesh는 서비스 코드를 변경할 필요 없이 분산된 기존 애플리케이션에 투명한 계층을 추가합니다. 마이크로서비스 간의 모든 네트워크 통신을 가로채는 환경 전반에 걸쳐 특별한 사이드카 프록시를 배포함으로써 서비스에 Red Hat OpenShift Service Mesh 지원을 추가합니다. 컨트롤 플레인 기능을 사용해 OpenShift Service Mesh를 구성하고 관리합니다.

OpenShift Service Mesh로 지원할 수 있는 활용 사례는 다음과 같습니다.

- 자동 mTLS로 서비스 간 통신에 대해 투명한 암호화 수행
- 예를 들어 여러 버전의 서비스를 제공하고 A/B 테스트 지원
- Kiali로 마이크로서비스의 통신 방식에 대한 관측성 확보
- Jaeger로 서비스 간 호출 추적

OpenShift의 다른 모든 플랫폼 기능과 마찬가지로 Service Mesh는 Red Hat 오퍼레이터로 설치됩니다.

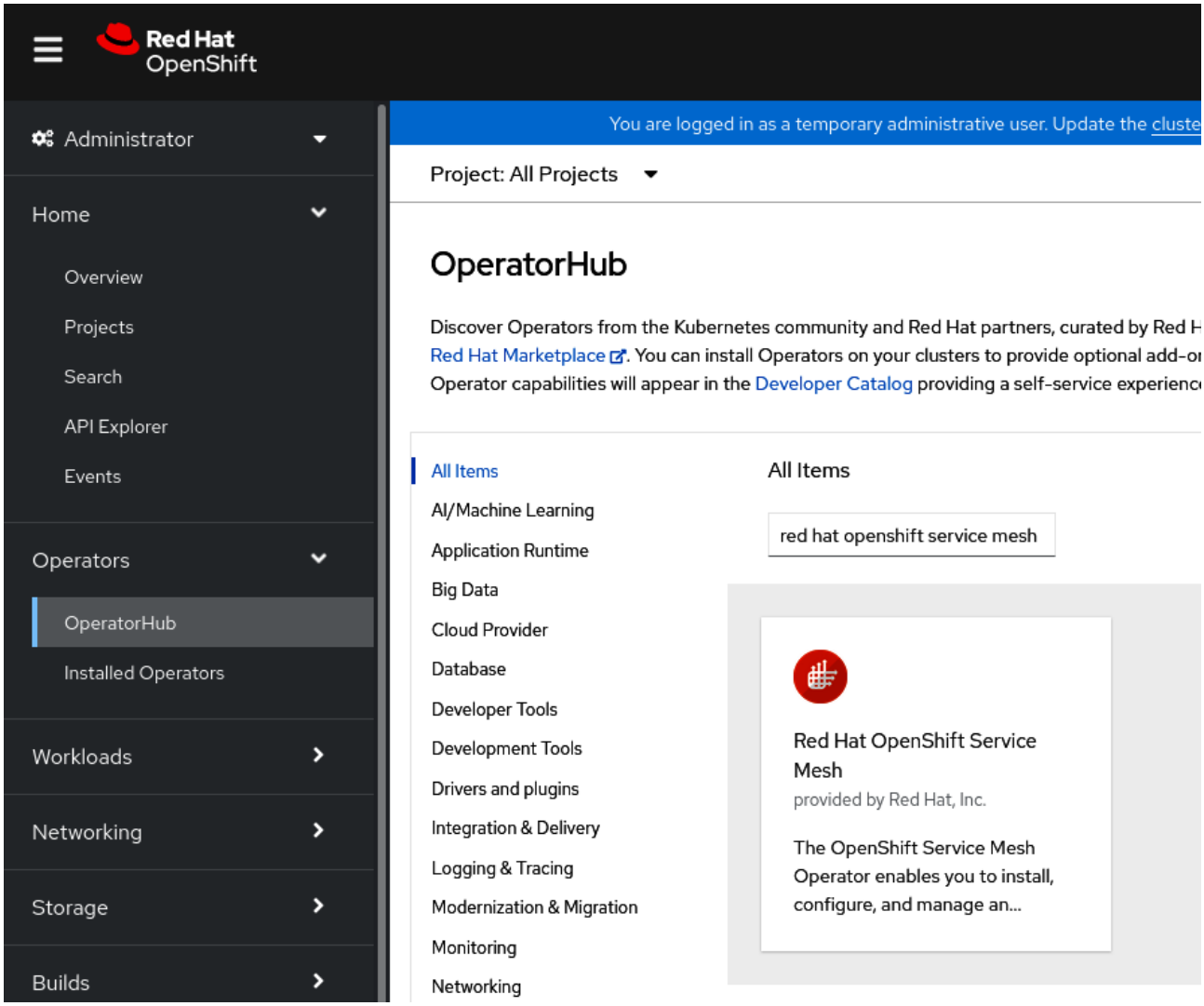


그림 8.13: OperatorHub를 통해 Service Mesh 설치하기

OpenShift Service Mesh 도큐멘테이션은 BookInfo 데모라고 하는 환상적인 예시 애플리케이션과 함께 제공됩니다. 이것은 서점을 에뮬레이션하는 단순한 애플리케이션으로서, OpenShift에서 실행됩니다.

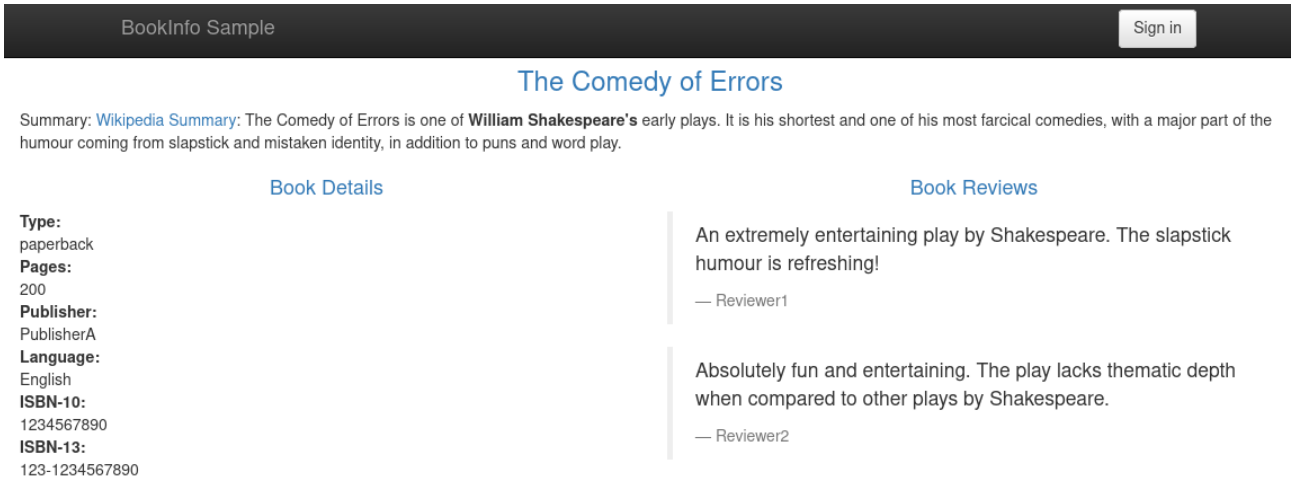


그림 8.14: BookInfo 애플리케이션

Service Mesh와 연동되는 BookInfo 애플리케이션을 확보하려면 Service Mesh 컨트롤 플레인을 생성해야 합니다. 다음과 같이 *그림 8.15*에 따라 OpenShift 그래픽 인터페이스를 통해 손쉽게 생성할 수 있습니다.

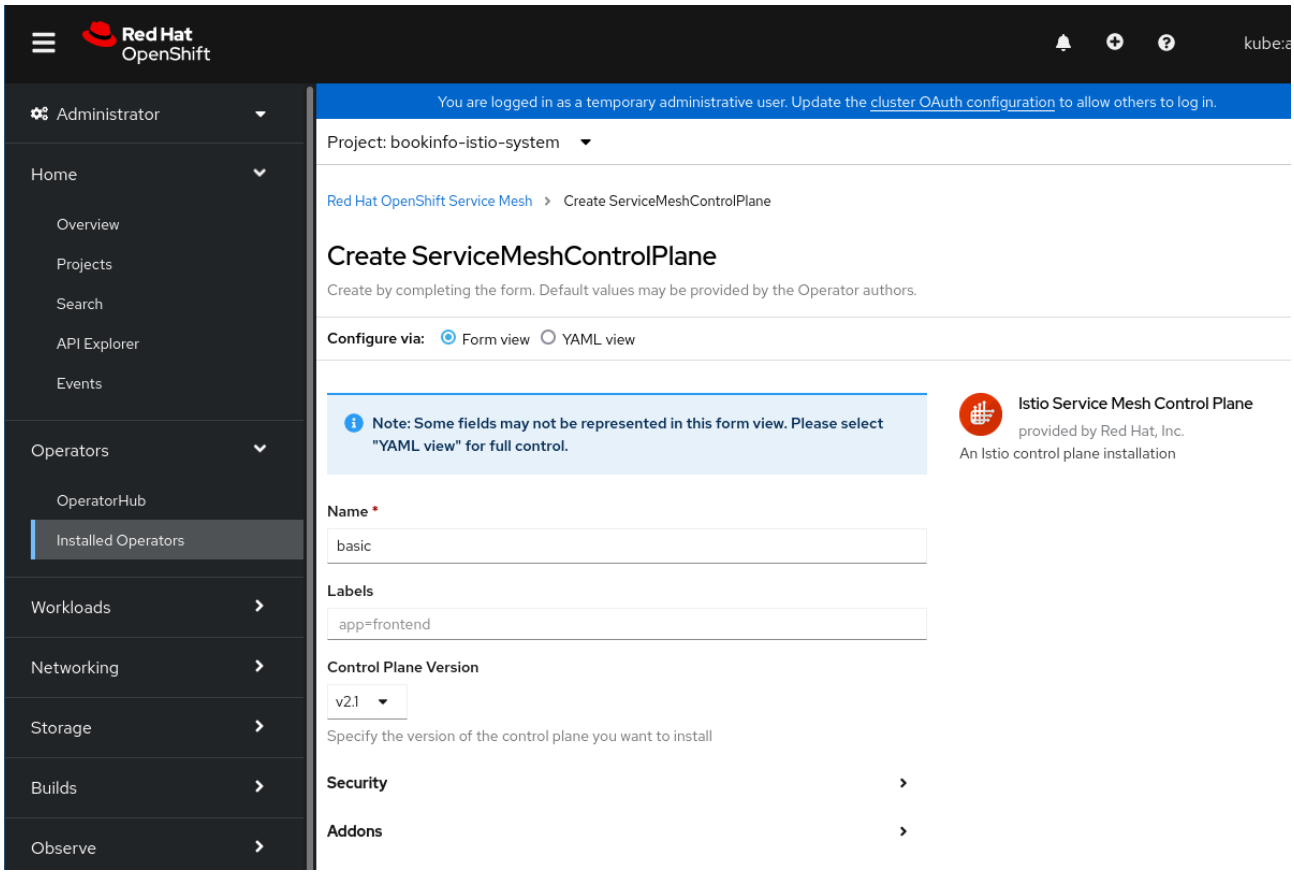


그림 8.15: BookInfo bookinfo-istio-system 프로젝트에서 컨트롤 플레인 설정하기

BookInfo 프로젝트는 컨트롤 플레인을 통해 인그레스 트래픽을 수용합니다. 이 경우 `bookinfo-istio-system`이라는 컨트롤 플레인을 수용하기 위해 별도의 프로젝트가 생성되었습니다.

Service Mesh 컨트롤 플레인과 프로젝트를 분리할 필요는 없으며, 여러 프로젝트에 걸쳐 Service Mesh 컨트롤 플레인을 공유하는 것도 가능합니다.

다음 두 섹션에서는 두 가지 Service Mesh 활용 사례, 즉 관측성과 분산 추적에 대해 자세히 알아보겠습니다.

플랫폼 서비스 – OpenShift Service Mesh – Kiali로 관측성 확보

이 애플리케이션을 구성하는 기본 포드를 보고 싶다면 Red Hat OpenShift 토폴로지 보기를 통해 이 애플리케이션이 여섯 가지 마이크로서비스로 구성되어 있음을 알 수 있습니다.

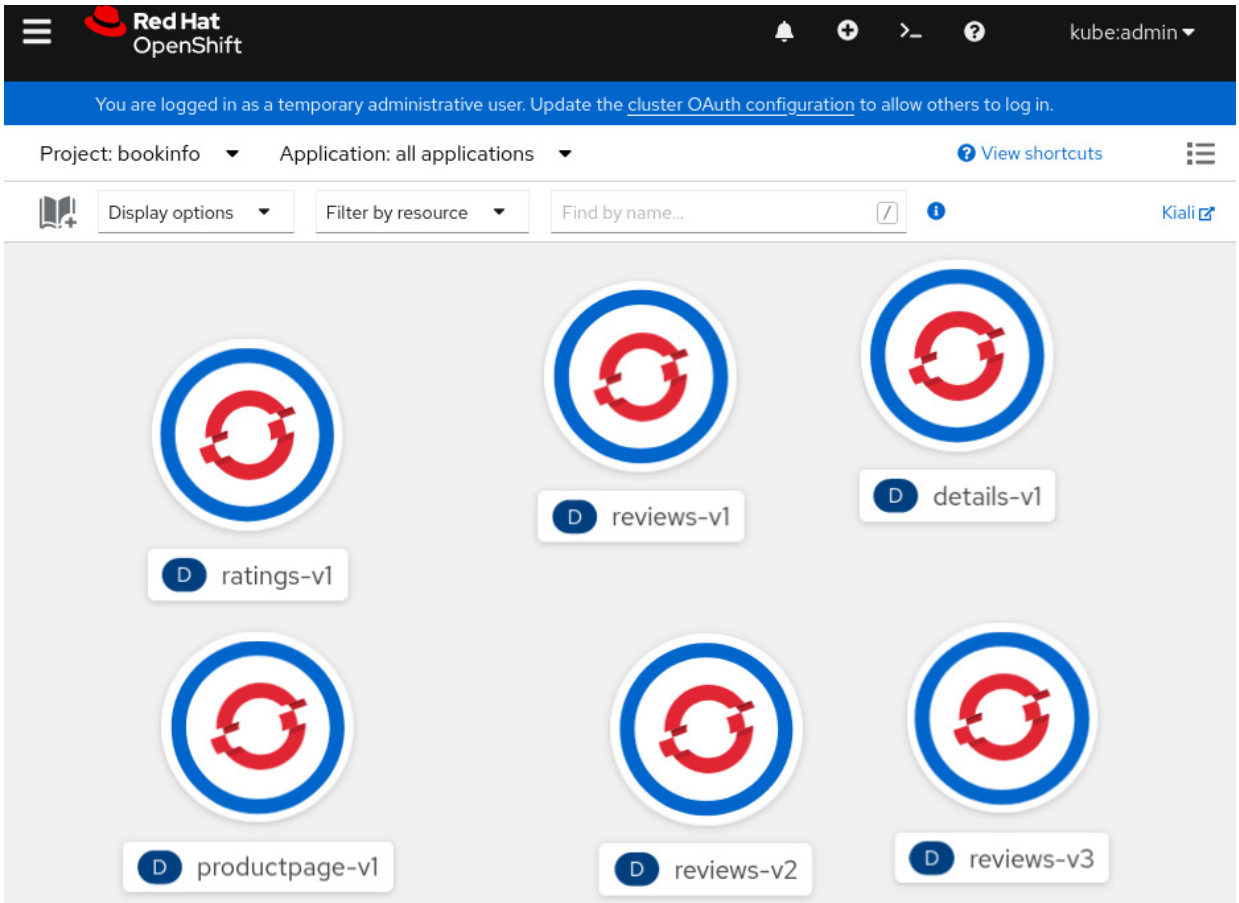


그림 8.16: BookInfo 애플리케이션을 구성하는 여섯 가지 서비스

이 보기가 유용하긴 하지만 이러한 여러 가지 서비스가 서로 어떻게 연결되어 있는지는 잘 알 수 없습니다. Service Mesh의 첫 번째 이점인 관측성을 보겠습니다. Service Mesh와 함께 제공되는 Kiali라고 하는 약간 다른 콘솔을 열면 이 여섯 가지 서비스의 아키텍처를 훨씬 더 정교하게 표현한 그림을 볼 수 있습니다.

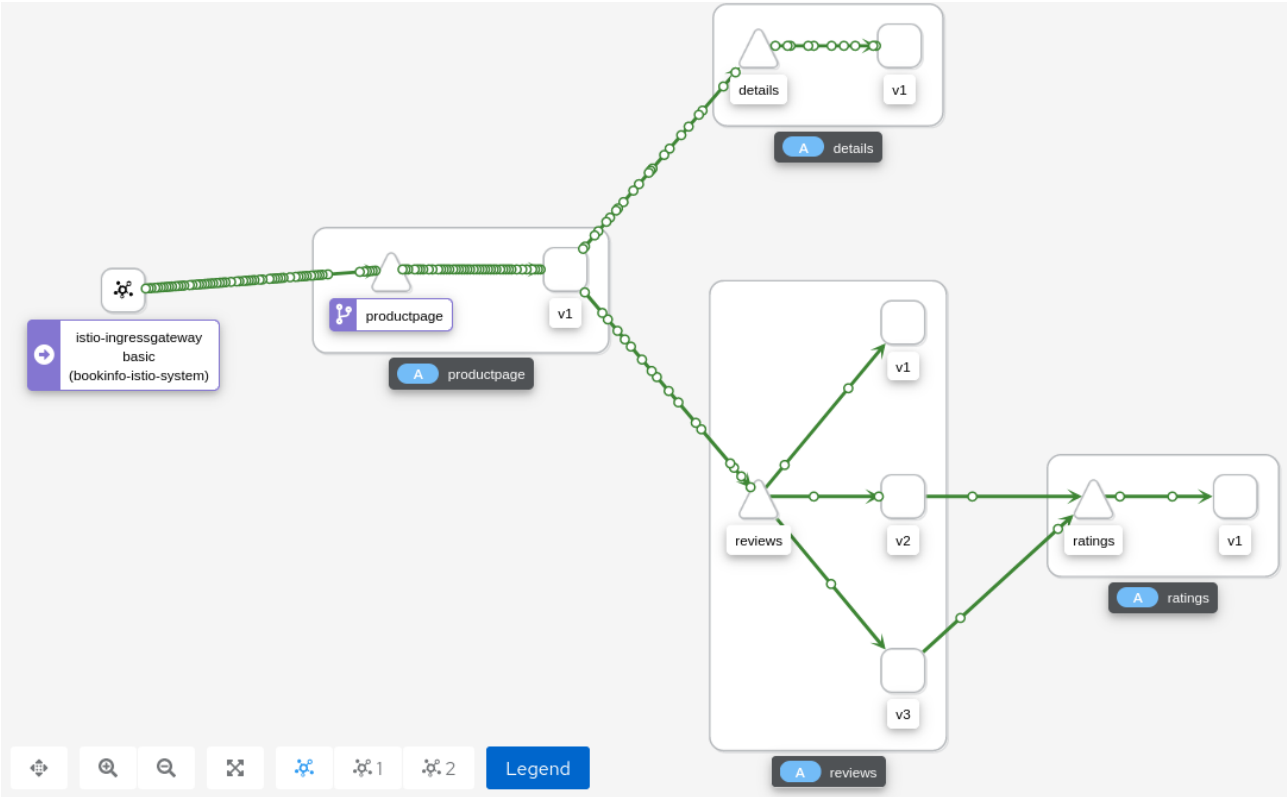


그림 8.17: Service Mesh의 지원을 받아 자동으로 생성된 애플리케이션 아키텍처 그래프

이 그림에서는 서비스 간 실제 연결성뿐 아니라 실시간 트래픽 다이어그램도 볼 수 있습니다. 이 경우 애플리케이션은 상당한 양의 트래픽을 수신하는데, 연결 상태에서 각 요청은 원형 애니메이션으로 표시되어 있습니다.

관리자는 이러한 관측성을 심화시켜 서비스 연결 중 하나를 클릭하여 트래픽 프로필을 볼 수 있습니다. 이 경우 트래픽은 대부분 **HTTP OK** 상태임을 알 수 있습니다.

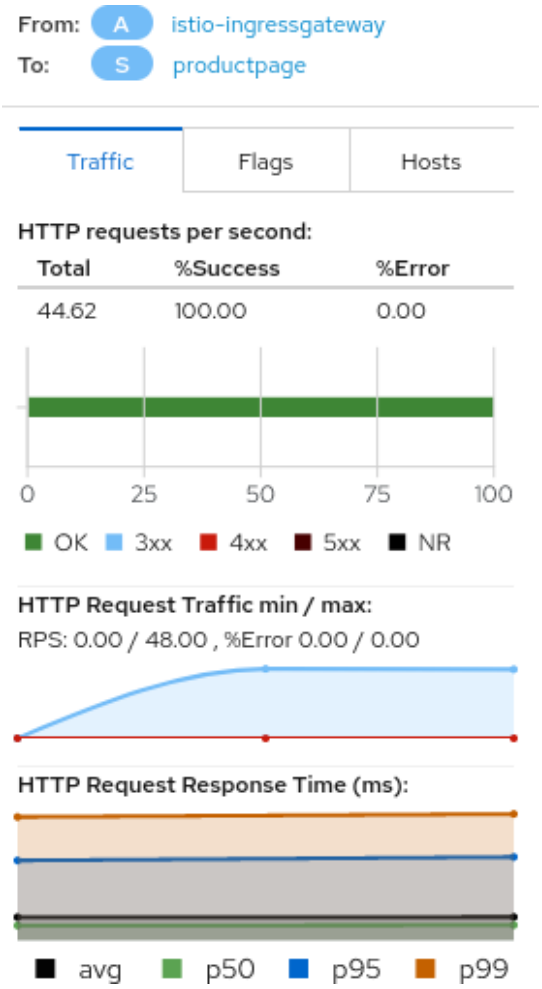


그림 8.18: 다양한 HTTP 상태

세부 정보 마이크로서비스를 종료하여 제로 복제본으로 축소함으로써 이 애플리케이션에 인공 오류를 도입할 수 있습니다.

그림 8.19: 세부 정보 서비스의 제로 복제본으로 장애를 시뮬레이션하기

이제 Kiali 보기로 돌아가면 몇 가지 구성 요소가 다이어그램에 추가되었음을 알 수 있습니다. 하지만 가장 두드러진 것은 세부 정보 서비스에 대한 연결이 오류를 표시하기 위해 빨간색으로 강조되어 있다는 것입니다.

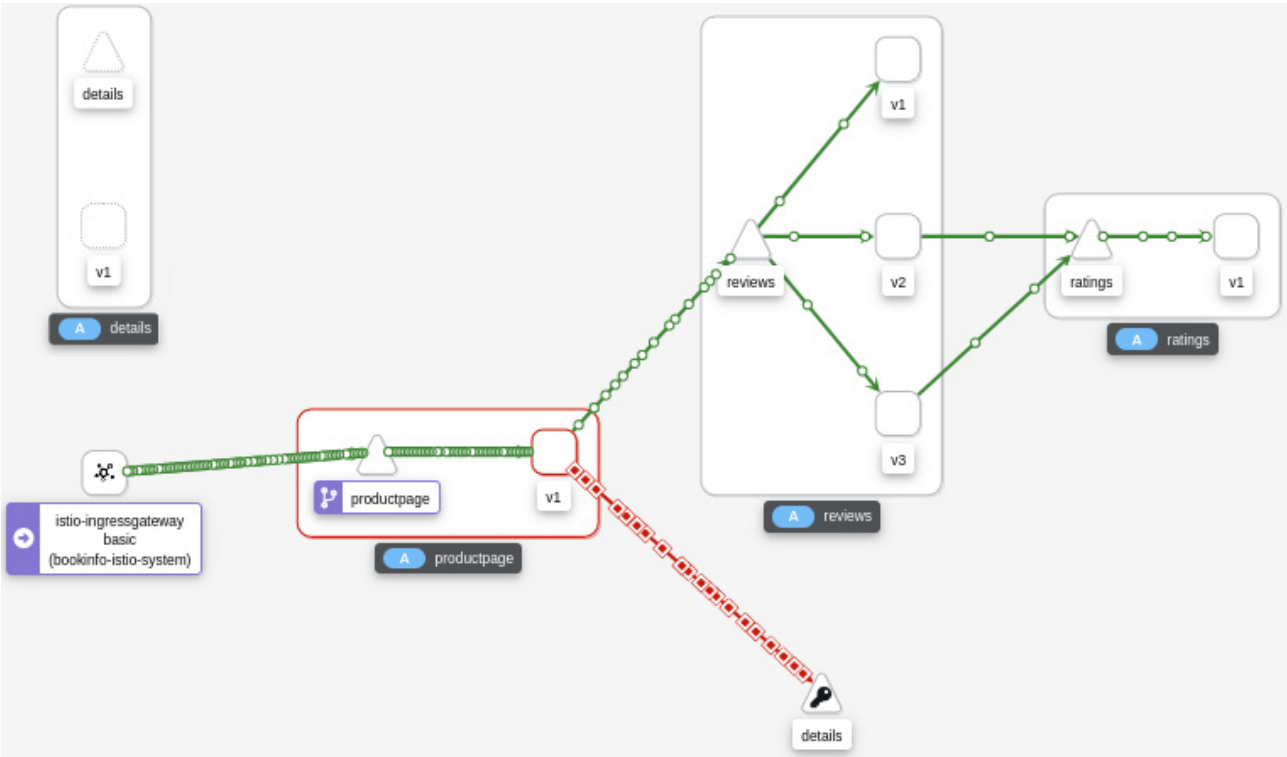


그림 8.20: 이 서비스에 문제가 있음을 보여주는 빨간색 연결

우리는 Service Mesh의 Kiali가 관측성 확보에 큰 도움이 된다는 것을 알게 되었습니다. BookInfo와 같은 소규모 애플리케이션에서는 Service Mesh가 유용합니다. 하지만 애플리케이션이 대형화되고 더 복잡해짐에 따라 때로 20개 이상의 마이크로서비스와 매우 다양한 유형의 상호 작용이 수반되므로 Service Mesh와 Kiali 같은 툴을 보유하는 것은 엄청난 가치가 있고 거의 필수적인 일입니다.

플랫폼 서비스 – Red Hat OpenShift Service Mesh – Jaeger를 이용한 분산 추적

Service Mesh가 제공하는, 엄청난 가치를 지닌 또 하나의 서비스는 Jaeger입니다. 이 서비스는 복잡한 마이크로서비스 애플리케이션에 대한 분산 추적을 지원합니다. 앞 섹션에서는 BookInfo 애플리케이션을 살펴보고 세부 정보 서비스가 오프라인 상태가 되면 요청이 실패하기 시작함을 알게 되었습니다.

이 경우 실패의 원인은 쉽게 파악할 수 있었습니다. 세부 정보 서비스를 사용할 수 없었기 때문입니다. 우리가 원인을 제공했죠! 하지만 원인이 무엇인지 이해하기 힘들어 추가 조사가 필요했다면 Jaeger의 분산 추적 기능이 유용했을 것입니다.

Jaeger는 시스템 전반의 후속 연결에서 첫 서비스의 요청을 추적합니다. Jaeger를 사용하려면 몇 가지 추가 애플리케이션 코드가 필요하지만 클라이언트 라이브러리와 최소 코드 변경으로 개발자는 비교적 수월하게 이 작업을 수행할 수 있습니다.

`productpage` 서비스(Python으로 작성됨)를 살펴보면 이 서비스에서 Jaeger 분산 추적을 지원하는 관련 코드를 구별할 수 있습니다. 이 코드는 다음과 같이 `productpage` 서비스에서 Jaeger 클라이언트 라이브러리를 가져옵니다.

```
from jaeger_client import Tracer, ConstSampler
from jaeger_client.reporter import NullReporter
from jaeger_client.codecs import B3Codec
```

클라이언트 라이브러리를 가져오고 나면 제품 페이지는 새로운 추적기를 설정해야 합니다. 이 코드는 다음과 같이 productpage 서비스에서 Jaeger Tracer를 초기화합니다.

```
tracer = Tracer(
    one_span_per_rpc=True,
    service_name='productpage',
    reporter=NullReporter(),
    sampler=ConstSampler(decision=True),
    extra_codecs={Format.HTTP_HEADERS: B3Codec()}
)
```

끝으로 다음과 같이 productpage 서비스에서 여전히 우리는 새로운 스팬(여러 서비스에 새로운 요청)을 생성하고 싶다고 Jaeger에게 알려줍니다.

```
span = tracer.start_span(
    operation_name='op', child_of=span_ctx, tags=rpc_tag
)
```

그러면 Jaeger는 여러 서비스 전반에서 이 추적기를 추적합니다. 이 서비스들은 Jaeger와 연동되도록 조정해야 하지만 클라이언트 라이브러리는 가장 널리 사용되는 프로그래밍 언어에 사용할 수 있습니다.

productpage 서비스의 전체 소스 코드는 GitHub에서 찾을 수 있습니다.

측정 코드에 대한 옵션은 현재 사용 중단된 것으로 간주되는 Jaeger 클라이언트 라이브러리를 사용하는 것과 OpenTelemetry 프로젝트의 오픈 표준 옵션을 사용하는 두 가지인데, 후자가 더 많이 사용됩니다.

- OpenTelemetry 라이브러리

애플리케이션 측정 작업이 완료되면 다음과 같은 추적이 표시됩니다.

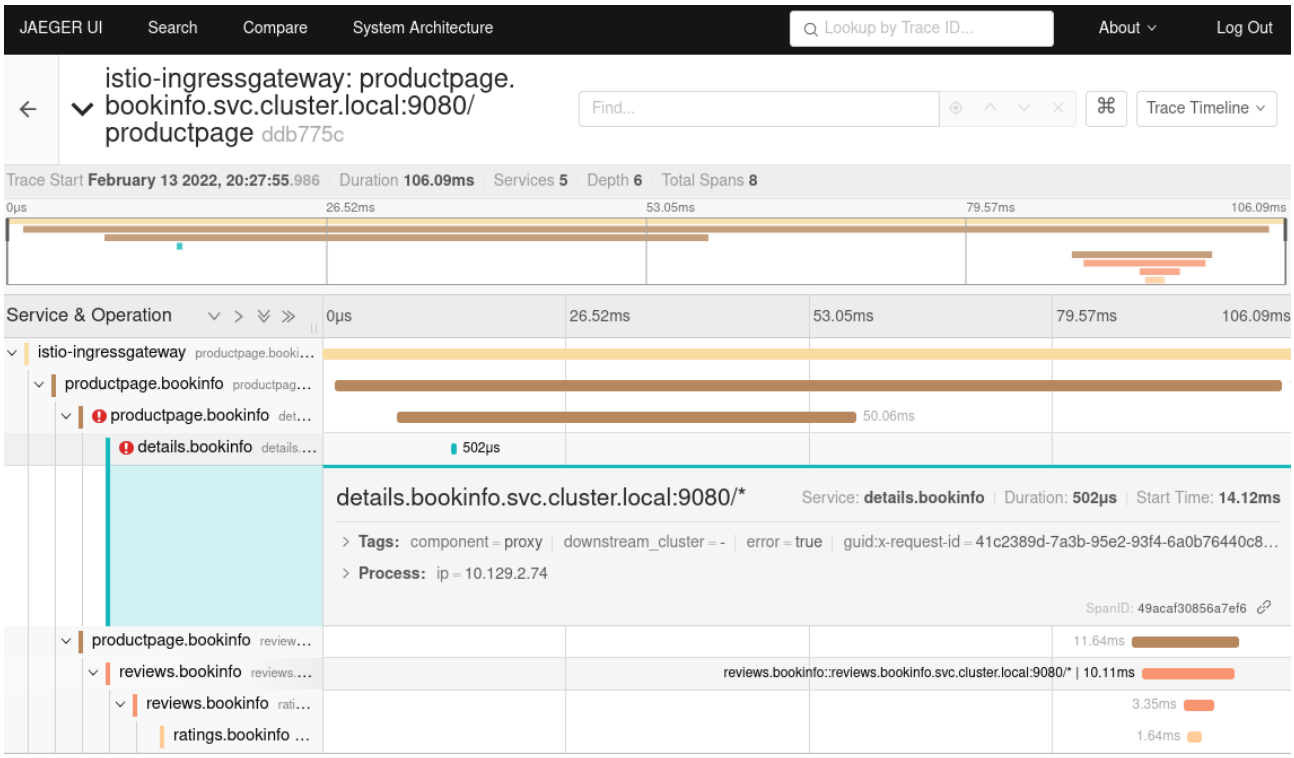


그림 8.21: 호출 추적

이 보기에서는 Kiali와 매우 유사하게 호출 추적을 보여주지만 여기에서는 계층형으로 표시되어 있습니다. 표시된 각 행을 확장하여 요청 기간, 시작한 시각, 소스 IP 주소, 기타 몇 가지 세부 정보를 자세히 표시할 수 있습니다. 이러한 수준의 정보는 복잡한 마이크로서비스 애플리케이션을 처리할 때 거의 필수적인 사항이 됩니다.

우리가 진단하고자 했던 오류를 살펴보면 추적 보기를 통해 세부 정보 마이크로서비스에 문제가 있음을 분명히 알 수 있습니다. 간단한 오류이지만 세부 정보 보기를 펼치면 이 서비스가 `HTTP 503` 오류 코드를 발행하고 있음을 알 수 있습니다.

그림 8.22: 오류 세부 정보

HTTP 503는 "서비스를 사용할 수 없음"에 해당하는 표준 오류 코드입니다. 이 경우 원인은 서비스가 제로 복제본으로 축소되었기 때문입니다. 서비스를 다시 확장하면 서비스가 복원됩니다.

Jaeger와 Kiali를 결합하면 마이크로서비스 애플리케이션이 더 복잡해짐에 따라 강력한 툴이 됩니다. 이 둘은 Azure Red Hat OpenShift 서비스의 일부로 제공되므로 추가 지출이나 서브스크립션이 필요 없습니다.

요약

이 장에서는 "비어 있는" 쿠버네티스 환경에 비해 애플리케이션 플랫폼인 Red Hat OpenShift가 제공하는 주요 부가 가치 서비스 몇 가지를 다루었습니다. OpenShift에서 해당되는 모든 업스트림 오픈소스 프로젝트를 설치함으로써 유사한 수준의 OpenShift 기능을 복제할 수 있지만 통합, 라이프사이클, 지원은 Azure Red Hat OpenShift에 수반되는 복잡성입니다.

이러한 다양한 플랫폼 서비스, 애플리케이션 서비스, 데이터 서비스, 개발자 서비스, 클러스터 서비스는 궁극적으로 개발자와 운영자가 쿠버네티스로 작업하고 여러 가지 복잡한 엔터프라이즈 애플리케이션을 배포할 때 이들의 생산성을 높입니다.

9장

다른 서비스와 통합

일반적으로 애플리케이션은 Red Hat OpenShift 내부에만 전적으로 존재할 수 없습니다. 대부분의 애플리케이션은 이런 저런 형식으로 외부 데이터베이스 또는 Azure 기반 서비스에 의존합니다.

Azure Red Hat OpenShift는 이 경우 어떤 종류의 연결성도 "훼손"하지 않습니다. 예를 들어 어떤 애플리케이션이 Azure CosmosDB에 의존한다면 애플리케이션을 변경하지 않고도 Azure Red Hat OpenShift에서 Azure CosmosDB로 연결할 수 있어야 합니다. 애플리케이션과 조직에 따라 이러한 외부 종속성 중 일부를 Azure Red Hat OpenShift에서 별도로 또는 동시에 배포할 수 있습니다.

애플리케이션으로 이러한 외부 종속성을 배포하는 것이 이익이 된다고 생각한다면 Azure Service Operator를 통해 이 프로세스를 크게 간소화할 수 있습니다. Azure CLI, Azure Cloud Shell 또는 ARM 템플릿 대신에 Azure Service Operator를 이용해 이러한 리소스가 마치 쿠버네티스 네이티브인 것처럼 배포할 수 있습니다.

Azure Service Operator

Azure Service Operator는 애플리케이션을 Azure Red Hat OpenShift에 배포하는 개발자나 관리자처럼 삶을 더 편하게 만들어 주는 또 하나의 오퍼레이터입니다. 한 가지 이점은 OpenShift 콘솔을 떠나지 않고도 Azure 서비스를 손쉽게 프로비저닝할 수 있게 해준다는 것입니다. 이를 통해 Azure CosmosDB와 같이 Azure 서비스 종속성이 있을 수 있는 애플리케이션을 더 빠르고 손쉽게 배포할 수 있습니다.

Azure Service Operator를 사용함으로써 얻을 수 있는 더 중요한 또 하나의 이점은 이러한 Azure 서비스 종속성을 표준 쿠버네티스 YAML 파일을 사용해 OpenShift 애플리케이션에 대한 정의와 함께 코드형 인프라(IaC) 접근 방식으로 저장할 수 있다는 것입니다.

이는 Azure Service Operator가 Azure에서 리소스의 정의에 부합하는 새로운 **CRD**(YAML로 정의됨)를 찾는 간단한 방식으로 작동합니다. Azure Service Operator는 이 YAML을 필수적인 Azure API 호출로 변환하여 Azure에서 요청되는 모든 것은 무엇이든 생성합니다. 오퍼레이터가 설치되어 있으면 사용자는 OpenShift Developer Catalog를 검색하여 Azure 공용 IP 주소, Azure SQL 데이터베이스 또는 Azure Firewall과 같은 여러 가지 일반적인 Azure 리소스에 대한 생성 화면을 볼 수 있습니다.

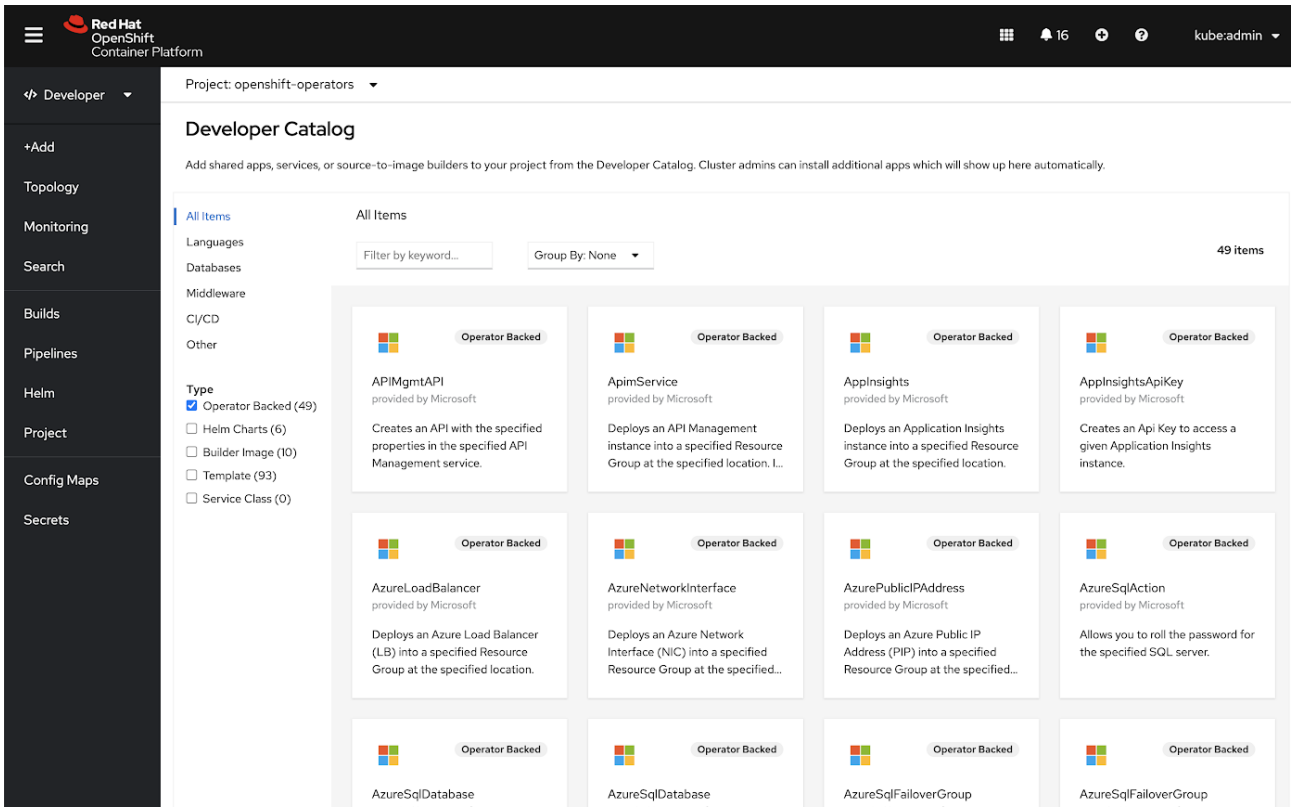


그림 9.1: Azure Service Operator가 노출한 일부 Azure 서비스

Azure Service Operator는 OperatorHub를 통해 설치할 수 있으며 모든 OpenShift 사용자에게 제공할 수 있습니다.

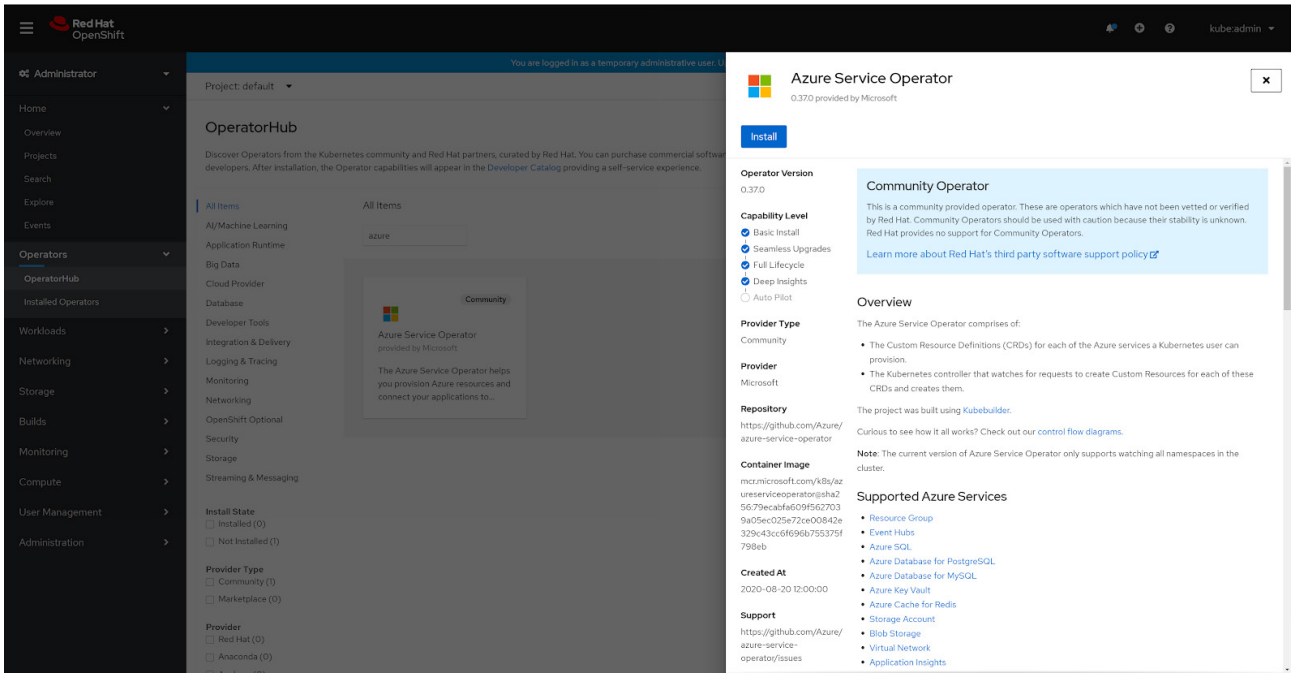


그림 9.2: OperatorHub 갤러리를 배경으로 한 Azure Service Operator 설치 화면

Azure에서 실행되는 서비스에 Azure Service Operator를 반드시 사용해야 하는 것은 아닙니다. 기본 Azure 서비스는 OpenShift가 아닌 Azure에 네이티브 방식으로 배포됨을 이해하는 것이 중요합니다. 하지만 Azure Service Operator를 이용하면 Azure 서비스 종속성이 있는 애플리케이션에 대해 이 작업을 더 빠르고 손쉽게 수행할 수 있습니다.

널리 사용되는 Azure Service Operator 활용 사례는 종속된 데이터베이스를 애플리케이션과 함께 배포하는 것입니다. 예를 들어 Azure CosmosDB의 인스턴스를 사용하는 웹 애플리케이션인 경우 OpenShift에 애플리케이션을 배포하는 과정의 일부로 Azure Service Operator를 이용해 Azure CosmosDB를 배포하세요. Azure Service Operator는 Azure SQL Database, Azure Database for MySQL, Azure PostgreSQL뿐 아니라 몇 가지 다른 리소스에 대한 지원도 포함합니다.

Azure Service Operator가 설치되어 있다고 가정하면 아래 쿠버네티스 매니페스트를 OpenShift 애플리케이션으로 저장하고 MySQL 데이터베이스 프로비저닝에 사용할 수 있습니다.

출처: Azure Service Operator 샘플 리포지토리에서 발췌함

```
apiVersion: azure.microsoft.com/v1alpha1
kind: MySQLServer
metadata:
  name: aso-wpdemo-mysqlserver
spec:
  location: eastus2
  resourceGroup: aso-wpdemo-rg
  serverVersion: “8.0”
  sslEnforcement: Disabled
  minimalTLSVersion: TLS10 # Possible values include: ‘TLS10’, ‘TLS11’, ‘TLS12’, ‘Disabled’
  infrastructureEncryption: Enabled # Possible values include: Enabled, Disabled
  createMode: Default # Possible values include: Default, Replica, PointInTimeRestore (not
implemented), GeoRestore (not implemented)
  sku:
    name: GP_Gen5_4 # tier + family + cores eg. - B_Gen4_1, GP_Gen5_4
    tier: GeneralPurpose # possible values - ‘Basic’, ‘GeneralPurpose’, ‘MemoryOptimized’
    family: Gen5
    size: “51200”
    capacity: 4
```

복잡한 종속성을 지닌 여러 Azure 서비스가 관련된 서비스에는 대개 Azure Service Operator를 사용하지 않습니다. 이러한 경우에는 Azure ARM 템플릿 또는 Bicep과 같은 툴이 더 적합할 수 있습니다.

Azure Service Operator에 대한 자세한 소개는 다음 블로그 포스트를 확인하세요.

- 블로그 포스트 – 2020년 9월
- OperatorHub.io 기반 Azure Service Operator
- GitHub 기반 Azure Service Operator

Azure DevOps와의 통합

많은 사용자가 Azure Red Hat OpenShift를 OpenShift Pipelines와 더불어 Azure DevOps와 통합하고자 할 것입니다. 이 솔루션들은 어떤 프로젝트에서 하나의 솔루션을 사용하고 그 외 프로젝트에서는 또 다른 솔루션을 사용하는 방식으로 조화롭게 공존할 수 있습니다. 또는 프로젝트에서 두 솔루션을 모두 사용할 때도 있습니다! 어느 솔루션을 언제 사용할지는 몇 가지 요인에 따라 결정됩니다.

일반적으로 말해 Azure DevOps는 다른 Azure 툴과 높은 수준에서 통합되지만 OpenShift뿐 아니라 다른 컴퓨팅 리소스에도 손쉽게 배포될 수 있습니다.

반면에 OpenShift Pipelines는 OpenShift와 함께 제공되는 효과적으로 통합된 제공 사항이며 일관된 멀티클라우드 경험을 제공합니다.

Azure DevOps는 다른 쿠버네티스 클러스터와 마찬가지로 OpenShift를 단순하게 처리합니다. 그러므로 모든 표준 쿠버네티스 인터페이스 및 API는 예상대로 작동합니다.

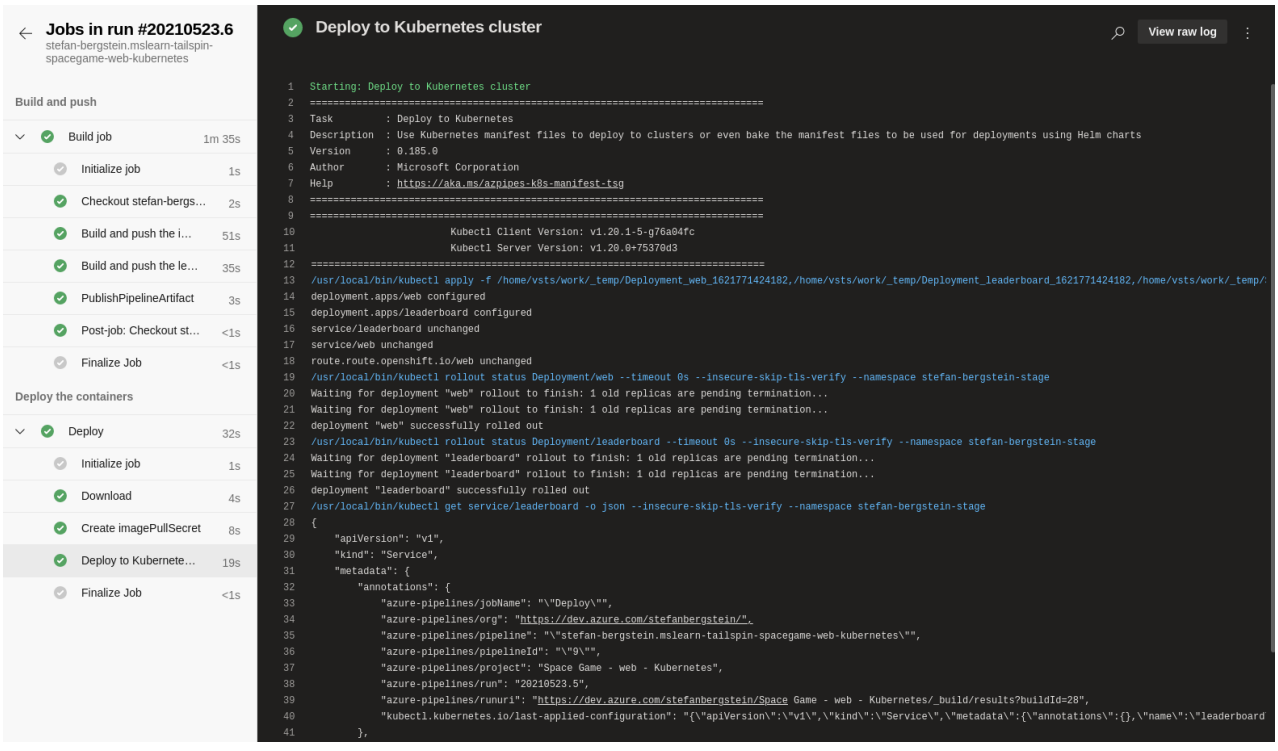


그림 9.3: OpenShift로 콘텐츠를 푸시하는 Azure DevOps 파이프라인

추가 자료

다음 커뮤니티 블로그 포스트에서는 Azure Red Hat OpenShift와 Azure DevOps를 시작하는 방법에 관한 훌륭한 튜토리얼을 제공합니다.

- 블로그 포스트 – 2021년 5월

Visual Studio Code와의 통합

OpenShift의 개발자 통합이 지닌 가치의 일부로 Visual Studio Code를 포함한 널리 사용되는 다수의 IDE를 위한 플러그인이 있습니다. 이를 통해 개발자와 관리자는 IDE를 떠나지 않고도 쿠버네티스와 OpenShift 리소스에 빠르고 쉽게 액세스할 수 있습니다.

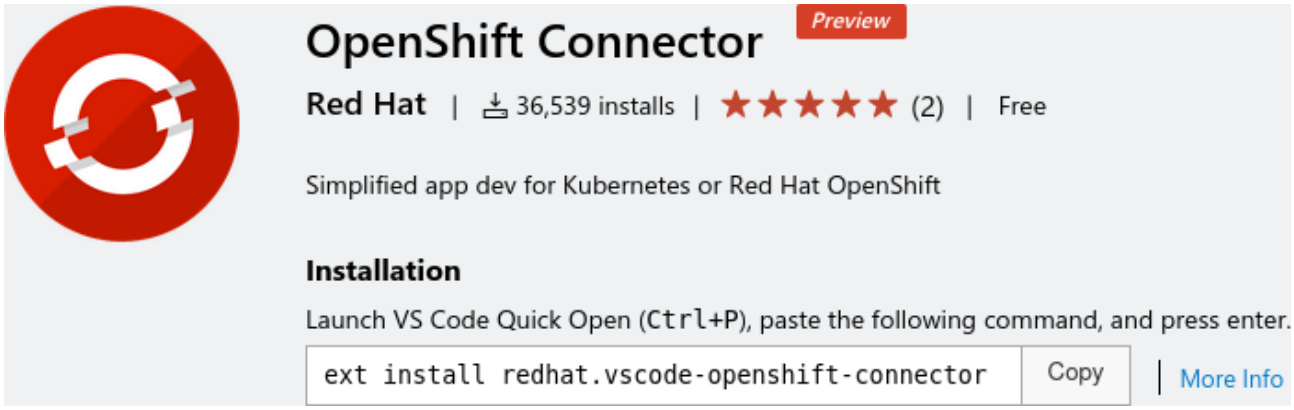


그림 9.4: OpenShift 커넥터 설치하기

Visual Studio Code에서 커넥터를 다운로드하여 설치하면 OpenShift 클러스터에 로그인하라는 메시지가 표시됩니다. 다음은 간단한 "hello world" 방식의 프로젝트로, Azure Red Hat OpenShift에 연결되어 있습니다.

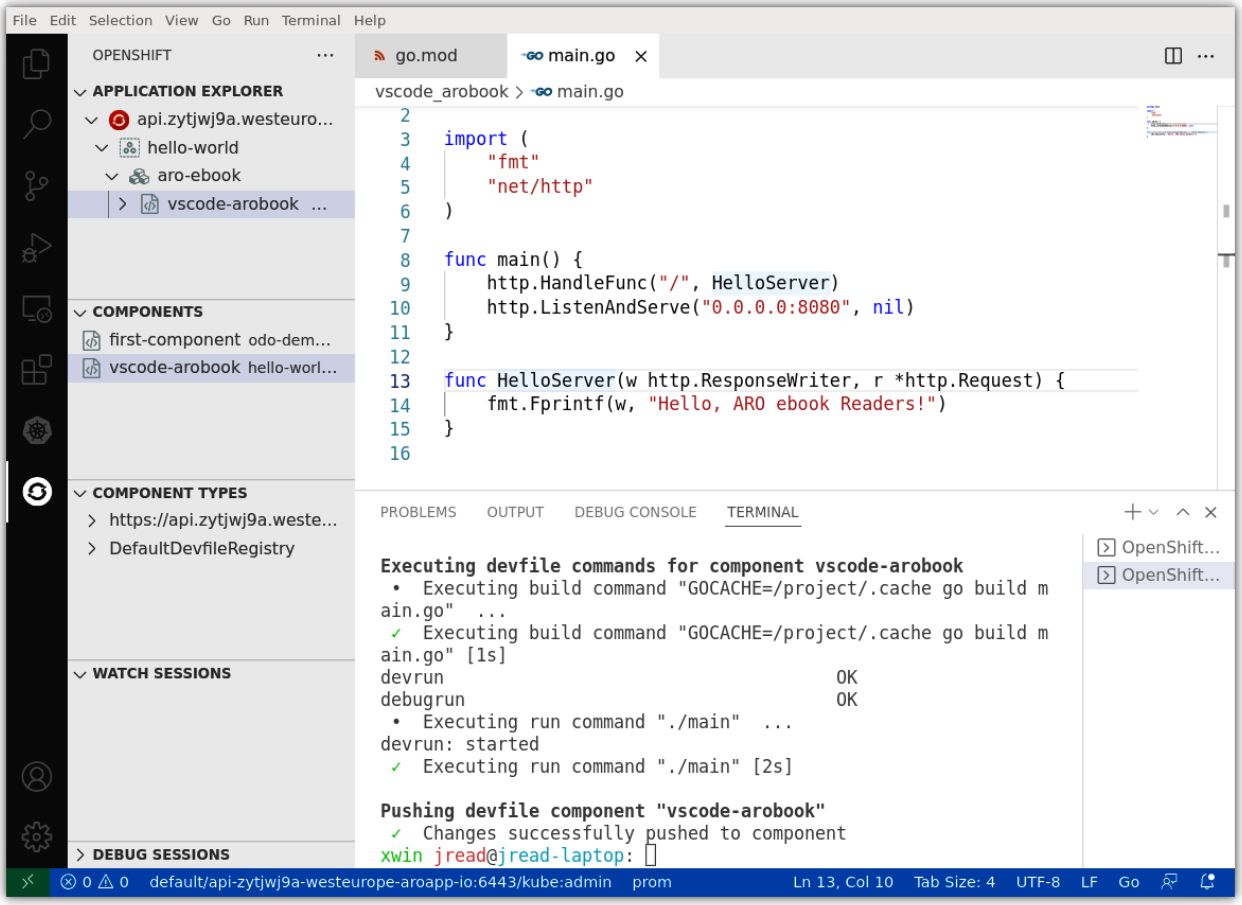


그림 9.5: Visual Studio Code OpenShift 커넥터로 배포된 Golang 프로젝트

OpenShift 커넥터를 Visual Studio Code와 함께 사용함으로써 누릴 수 있는 이점 중 하나는 일반적으로 "odo"라고 하는 널리 사용되는 OpenShift 툴인 "OpenShift Do"에 그래픽 프론트엔드를 사용할 수 있다는 것입니다. 따라서 개발자는 원시 쿠버네티스 구성 요소보다 높은 수준의 개념으로 사고할 수 있습니다. 개발자는 Deployments, ReplicaSets 등에 관해 세부적으로 신경 쓰지 않아도 됩니다. 위 스크린샷을 보면 이것이 HTTP 서비스가 노출된 간단한 프로젝트임을 알 수 있습니다.

또한 커넥터는 소스 제어 기능을 먼저 사용할 필요 없이 코드 변경 사항을 OpenShift로 직접 푸시할 수 있는 기능을 포함합니다. 이 기능은 소스 제어 기능을 매번 푸시하여 새 컨테이너를 빌드하고 배포할 필요가 없어 신속한 개발에 매우 유용합니다.

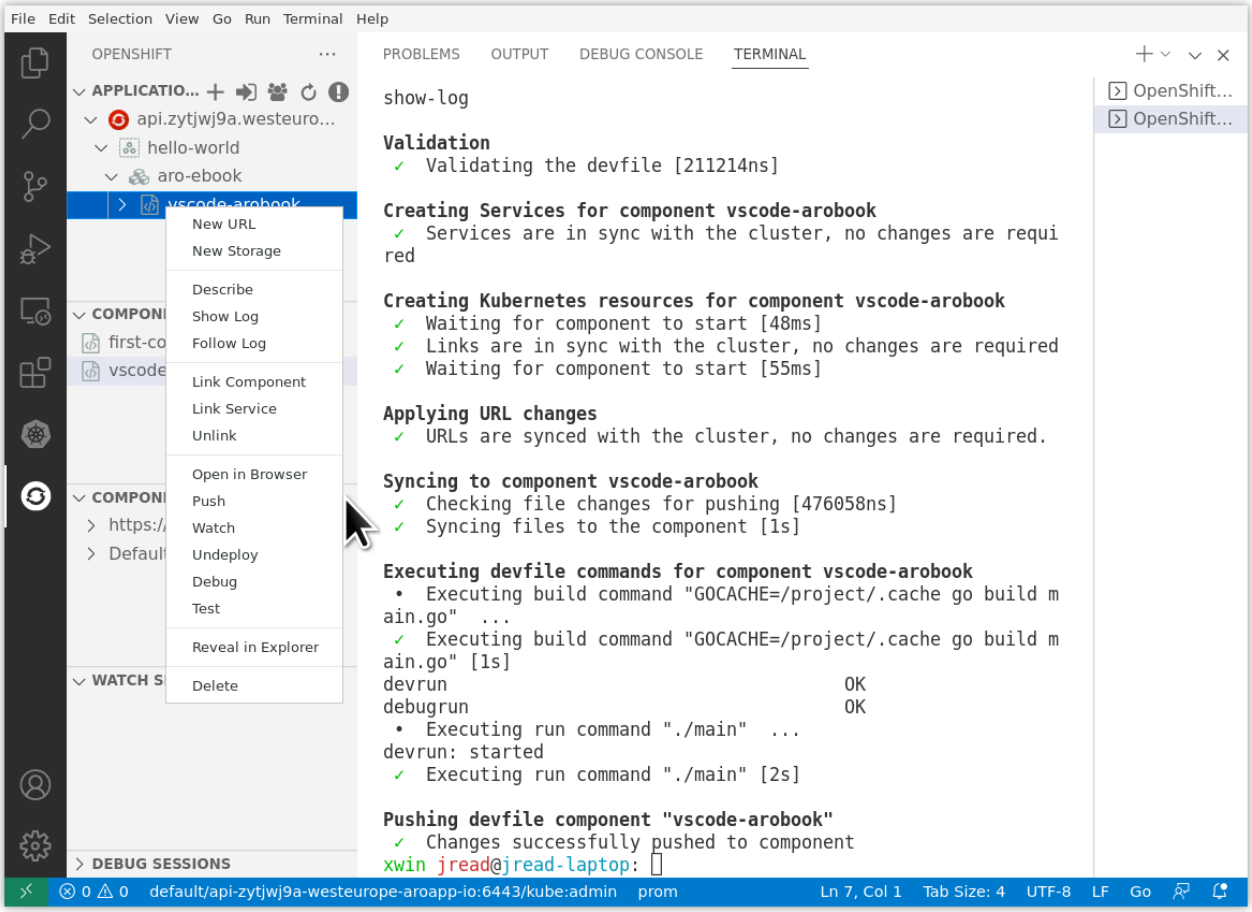


그림 9.6: 프로젝트의 "푸시" 메뉴와 터미널 창의 최근 푸시

이 커넥터는 Visual Studio Code 사용을 선호하는 개발자를 위해 개발 및 테스트 주기를 가속화합니다.

그림 9.7: OpenShift에서 실행되는 기본 Golang 애플리케이션

물론 개발자는 Visual Studio Code만 사용하도록 제한되지 않습니다. 많은 개발자가 Vim, Eclipse, 기타 편집기로 작업하지만 Visual Studio Code는 "경량화 IDE" 스타일의 경험을 좋아하는 개발자에게 매우 인기가 많습니다.

추가 자료

- 데모 동영상 – Visual Studio Code를 OpenShift와 함께 사용하기
- Visual Studio Code OpenShift 커넥터

GitHub Actions와의 통합

이제 GitHub Actions를 사용해 Azure Red Hat OpenShift를 포함한 모든 Red Hat OpenShift 환경에 배포할 수 있습니다. GitHub 리포지토리에서 **작업** → **새 워크플로우**로 이동하여 사용 가능한 작업 목록에서 **OpenShift**를 선택합니다.

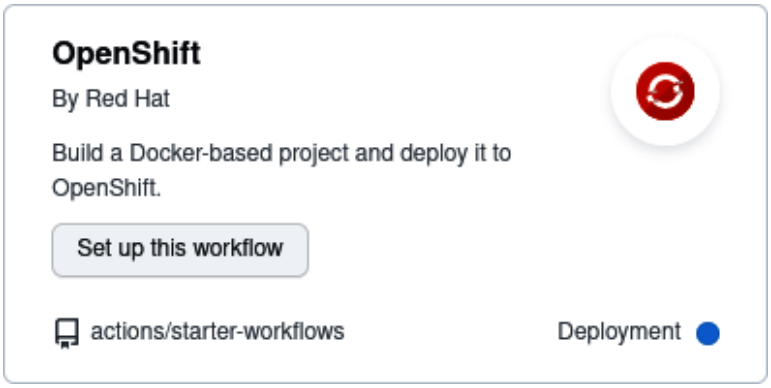


그림 9.8: GitHub에서 OpenShift로 배포하기

기본 템플릿은 다음과 같이 OpenShift 클러스터에 연결하도록 설정된 몇 가지 환경 변수만 필요한 일련의 탁월한 예시 작업과 함께 제공됩니다.

```
name: OpenShift

env:
  # ✎ EDIT your repository secrets to log into your OpenShift cluster and set up the context.
  # See https://github.com/redhat-actions/oc-login#readme for how to retrieve these values.
  # To get a permanent token, refer to https://github.com/redhat-actions/oc-login/wiki/Using-a-
Service-Account-for-GitHub-Actions
  OPENSHIFT_SERVER: ${{ secrets.OPENSHIFT_SERVER }}
  OPENSHIFT_TOKEN: ${{ secrets.OPENSHIFT_TOKEN }}
  # ✎ EDIT to set the kube context's namespace after login. Leave blank to use your user's
default namespace.
  OPENSHIFT_NAMESPACE: ""
  ...
```

이 작업을 수행하는 방법을 설명하는 훌륭한 블로그 포스트와 데모 동영상은 다음과 같습니다.

- 블로그 게시물
- GitHub Actions와 OpenShift 데모 동영상

요약

이 장에서는 고객이 Azure Red Hat OpenShift와 함께 사용하는 것으로 알고 있는 여러 가지 일반적인 통합에 대해 알아보았습니다. 이 장의 도입부에서 기술한 것처럼 OpenShift API를 통해 이 장에서 나열되지 않은 더 많은 서비스와 통합할 수 있습니다. OperatorHub에 나열된 오퍼레이터를 사용해 엄청난 수의 서비스를 손쉽게 통합할 수도 있습니다.

다음 장에서는 애플리케이션 팀을 온보딩하고 조직 내에서 서비스 도입 시 견인력을 얻는 방법에 관한 몇 가지 모범 사례와 권장 사항을 함께 알아보겠습니다.

10장

워크로드 및 팀 온보딩

프로비저닝 전 단계를 모두 실행하고, Azure Red Hat OpenShift를 프로비저닝하고, 필요한 프로비저닝 후 단계를 모두 실행했다면 이제 개발자와 애플리케이션을 지원하고 클러스터로 온보딩하기만 하면 됩니다. 이를 어떻게 시작할 것인가는 조직 문화에 달려 있습니다. 어떤 개발자와 애플리케이션 팀은 "심층적으로 자세히 알아본" 후에 시작하고 싶어 하는 반면, 어떤 팀은 더 많은 지침을 제공하면 이를 고마워할 수 있습니다.

이 장에서는 개발자와 애플리케이션 팀이 Azure Red Hat OpenShift를 사용할 때 더 신속하게 시작하고 실행할 수 있도록 보장하는 훌륭한 기반이 되는 일련의 체크리스트를 제공합니다.

이 체크리스트를 조직의 구조와 문화에 맞게 조정하는 과정에서 내용을 확장하는 것이 좋습니다.

체크리스트: 온보딩 전

새 워크로드를 온보딩하기 전에 Azure Red Hat OpenShift가 조직의 모범 사례에 적합한 방식으로 배포 및 설계되었다는 것을 애플리케이션 팀 또는 해당 워크로드 소유자에게 알려주는 것이 중요합니다.

- Azure Red Hat OpenShift는 조직의 애플리케이션에 대해 승인된 Azure 랜딩 구역 또는 다른 Azure 환경으로 배포되었습니다.
- 클러스터에는 스토리지 클래스, 인증과 같은 "2일 차" 설정을 모두 완료해야 합니다*(6장: 프로비저닝 전 - 2일 차*에 기술되어 있음).
- 클러스터는 이미 플랫폼 팀에 의해 모두 구성되어 있으며 플랫폼 팀의 지원을 받습니다. 또한 Red Hat과 Microsoft가 제공하는 통합 지원을 받습니다.
- 클러스터는 사용 가능하며 안정적인 최신 버전으로 업데이트되었으며 패치가 잘 적용되어 있습니다. 앞으로도 계속해서 적절한 패치가 적용될 것입니다.

- Azure Red Hat OpenShift 클러스터는 다음과 같은 필수 리소스에 연결되어 있습니다.
  - Azure ExpressRoute 또는 VPN을 통해 온프레미스 환경에 연결
  - 엔터프라이즈 아티팩트 리포지토리(예: Java Maven 리포지토리)에 연결
  - 애플리케이션 빌드 중에 종속성을 다운로드하기 위해 퍼블릭 인터넷에 연결

파일럿 워크로드 온보딩

Azure Red Hat OpenShift를 조직에 롤아웃하는 일반적인 패턴은 최소 두 개의 파일럿 워크로드를 온보딩하는 것입니다.

- **의미 있는 최소 워크로드** – SRE 팀이 가장 이상적인 첫 번째 파일럿 워크로드는 기술 요구 사항이 매우 단순한 소규모의 잘 알려진 애플리케이션입니다. 이 애플리케이션은 일반적으로 조직 내에 실제 비즈니스 소유자가 있어야 한다는 점에서 단순한 "Hello World" 애플리케이션보다 약간 상위에 있습니다. 하지만 첫 번째 워크로드의 경우 Azure Red Hat OpenShift 클러스터가 클러스터 수준의 비즈니스 요구 사항을 충족할 수 있는지 확인하는 데 강조점이 주어져 있습니다. 이러한 요구 사항은 Azure 연결성, Azure Active Directory 로그인, "가장 쉬운 작업"의 간편한 식별 등 모든 애플리케이션에서 겪을 수 있는 일반적인 문제입니다.
- **의미 있는 참조 워크로드** – 단순한 최소 워크로드가 배포되고 나면 두 번째 워크로드가 충분히 복잡하고 의미 있는 경우 도입을 진전시키는 데 큰 도움이 됩니다(비즈니스에 중요하거나 비즈니스 크리티컬함). 이 워크로드는 중요한 데이터베이스에 대한 연결성, 성능 테스트, 로깅/지표와 같은 문제를 대규모로 식별하고 해결하는 데 도움이 됩니다. 중요한 점은 이 의미 있는 참조 워크로드를 조직 내에서 **내부 참조**로 사용할 수 있다는 것입니다. 이로써 향후 개발 및 애플리케이션 팀은 Azure Red Hat OpenShift 플랫폼에 대한 확신을 얻을 수 있습니다.

물론 조직에 효과적일 수 있는 몇 가지 첫 워크로드를 온보딩하는 다른 패턴이 있습니다. 곧 제거될 데이터센터의 애플리케이션을 대상으로 하거나 기존 Java 애플리케이션 서버에서 실행되는 애플리케이션을 대상으로 해야 할 수도 있습니다.

체크리스트: 추가 팀과의 온보딩 회의

새로운 개발 또는 애플리케이션 팀을 클러스터로 온보딩할 때 온보딩 회의를 호스팅하는 것이 좋습니다.

- 팀이 사용할 한 개 또는 일련의 네임스페이스를 생성합니다.
- 팀을 대상으로 짧은 Red Hat OpenShift 데모를 수행하여 'GitHub에서 배포하기'와 같은 주요 기능(또는 이와 동등한 기능)을 알려줍니다.
- 클러스터에 대상 워크로드(CPU, RAM, 스토리지 등)를 배포할 수 있을 만큼 예비 용량이 충분한지 확인합니다.
- 팀 구성원이 클러스터에 로그인할 수 있는지 확인합니다. Azure Active Directory와 연결하면 더 쉽게 확인할 수 있습니다.
- 클러스터를 관리하는 SRE 팀(또는 이와 유사한 팀)에 연락처 세부 정보를 제공합니다.
- 다음과 같은 OpenShift 시작하기 링크 목록을 제공합니다.
    - http://learn.openshift.com
    - https://github.com/openshift-labs/starter-guides
    - http://docs.openshift.com

체크리스트: 정기적인 상태 점검 호출

개발 또는 애플리케이션 팀이 클러스터 배포를 시작한 후에는 많은 경우 정기적인 상태 점검 회의를 갖는 것이 좋습니다. 이 회의는 일일 스탠드업의 일부가 될 수 있습니다. 또는 매주 30분 주기만으로 충분할 수 있습니다. 확인할 사항은 다음과 같습니다.

- 팀이 전반적으로 어떻게 해나가고 있는가? 배포된 애플리케이션이 있는가?
- 클러스터 사용과 관련해 최근에 문제가 발생한 적이 있는가(Red Hat OpenShift에 관한 지식 또는 연결성 문제)?
- 경험에 부정적인 영향을 미친 Azure 또는 클러스터 중단 사태가 있었나?
- SRE 팀 가용성에 대한 알림과 질문에 답하기 위한 연락처 세부 정보

유사한 프로젝트에 정기적인 상태 점검 호출을 진행할 때 이미 사용했을 수 있는 다른 것이 무엇인지 생각해 보세요. 체크리스트에 효과적일 것으로 생각되는 것을 추가합니다.

안티패턴: 개발자 플레이그라운드/샌드박스

조직의 개발자 채택을 장려하기 위한 일반적인 "안티패턴"은 흔히 "개발자 플레이그라운드"라고 하는 샌드박스 유형의 환경을 제공하고 개발자와 애플리케이션이 Azure Red Hat OpenShift 도입을 시작하기를 기대하는 것입니다. Azure Red Hat OpenShift는 후속 지원 또는 리소스를 추가하지 않으면 흡수하기에 너무 많은 복잡성을 지닌 위협적인 환경이 될 수 있습니다. 대부분의 경우 "샌드박스" 패턴을 도입하면 클라우드 컴퓨팅 경비가 낭비되고 클러스터가 빈 상태가 될 수 있습니다.

하지만 조직이 과거에 개발 팀과 "샌드박싱"을 진행한 경험이 있는 경우 이를 최대한 성공으로 이끌기 위해 시도할 수 있는 몇 가지 팁은 다음과 같습니다.

- Azure Red Hat OpenShift가 할 수 있는 일에 관한 짧은 데모를 제공하세요.
- 몇 가지 도큐멘테이션과 앞서 간단히 살펴본 시작하기 링크를 제공하세요.
- "해커톤" 이벤트를 주최하여 소규모 경쟁 형식으로 OpenShift를 이용해 무언가를 빌드하도록 개발자에게 과제를 부여하세요.
- 확장된 지원, SRE 팀 소개, 더 자세한 안내가 지원되는 온보딩 경험(옵션)을 제공하세요.

앞서 간략히 살펴보았듯이 안내가 제공되는 도입 패턴 체크리스트를 따르면 플랫폼 도입이 성사되는 것을 볼 가능성이 더 높습니다.

Azure Red Hat OpenShift 개발자 워크숍

Azure Red Hat OpenShift 개발자 워크숍(https://aroworkshop.io)은 사전 생성된 유용한 워크숍으로서, 조직 내에서 사용해 Azure Red Hat OpenShift에 대한 핸즈온 경험을 안내와 함께 제공할 수 있습니다. 이 워크숍은 두 가지 랩 실습으로 구분되어 있으며, 둘 다 OpenShift를 처음 사용하는 개발자 또는 오퍼레이터를 튜토리얼을 통해 안내합니다. 두 랩 모두 주요 OpenShift 기능을 강조하고 이 기능을 사용하는 방법을 보여줍니다. 랩 1은 현대적인 마이크로서비스 설계에 대한 소개이고, 랩 2는 서비스의 세부 내용을 살펴봅니다.

조직 내에서 Azure Red Hat OpenShift에 대한 인지도를 높이는 데 유용한 활동은 내부 Azure Red Hat OpenShift 클러스터에서 aroworkshop.io 콘텐츠를 사용하는 것입니다. 워크숍 전 논의에서는 조직의 기존 Azure Red Hat OpenShift 서브스크립션 내에서 Azure Red Hat OpenShift가 배포되어온 방식을 설명할 수 있습니다. 워크숍 중에 이 서비스를 사용하는 것은 프로덕션 애플리케이션 호스팅을 위해 Azure Red Hat OpenShift를 사용하는 것과 유사한 경험이 될 수 있습니다.

요약

이 장에서는 워크로드와 팀을 Azure Red Hat OpenShift로 온보딩할 수 있는 몇 가지 유용한 체크리스트와 지침을 제공했습니다. 일반적인 안티패턴을 지원이 제공되지 않는 "샌드박스" 클러스터로 기술했습니다. 모든 조직에 적용될 리소스 및 체크리스트 항목의 전체 목록을 설명하는 것은 어느 지침에서도 어려운 일이므로 요구 사항에 맞게 이 장의 내용을 조정하는 것이 중요합니다.

클라우드는 다수의 매력적인 서비스를 제공합니다. 하지만 많은 서비스가 간과되거나 잘 도입되지 않는 이유는 조직이 기술과 기술 배포 방식에 너무 초점을 맞추기 때문입니다. 이러한 주제들을 이해하는 것이 중요하지만 이 서비스를 프로덕션과 효율적인 도입 단계로 유도하는 것과 관련해서는 문제의 일부일 뿐입니다. 이 장에서는 교육, 정기 체크인, 그리고 이와 유사한 활동이 Azure Red Hat OpenShift 도입을 더 성공적으로 수행하는 데 필요한 이유를 강조하고자 했습니다.

11장

결론

개발 및 운영 팀은 클러스터와 CI/CD 파이프라인을 프로비저닝, 설정, 유지 관리, 감독하는 작업에 대부분의 근무 시간을 보낼 수 있습니다. 이 경우 소중한 시간을 자신이 가장 잘 하는 일, 즉 애플리케이션을 최첨단 상태로 유지하는 일에 온전히 사용할 수 없습니다.

이 가이드에서 학습한 것처럼 Azure Red Hat OpenShift를 이용하면 완전 관리형 Red Hat OpenShift 클러스터를 배포하는 과정에서 이 클러스터를 실행하는 데 필요한 인프라를 빌드 또는 관리하느라 염려하지 않아도 됩니다. 쿠버네티스를 단독으로 실행할 경우 주로 Azure Red Hat OpenShift로 자동화할 수 있는 태스크에서 신경 써야 할 몇 가지 핸즈온 관련 주의 사항이 있음을 확인했습니다.

조직을 위해 어떤 클러스터 관리 전략을 선택할지 판단할 때 쿠버네티스 유형의 플랫폼과 쿠버네티스 프레임워크에 구축되고 탁월한 추가 장점을 제공하는 Azure Red Hat OpenShift를 통해 얻게 될 장점과 단점을 고려하시기 바랍니다.

Azure Red Hat OpenShift에 대해 자세히 알아보려면 제품 페이지를 방문하거나 도큐멘테이션 섹션을 확인해 보세요. 핸즈온 워크숍에 참석하고 등록을 통해 편한 시간에 웨비나를 시청할 수도 있습니다. Microsoft와 Red Hat에 연락하여 Azure Red Hat OpenShift 평가 관련 지원을 받으실 것을 적극 권장합니다.

저자 및 버전

James Read <james@redhat.com>

Red Hat 수석 솔루션 아키텍트,

Microsoft 관련 내용 – AROv4용으로 업데이트 및 개정

Ahmed Sabbour <asabbour@microsoft.com>

Microsoft 선임 제품 마케팅 매니저,

Azure Red Hat OpenShift 관련 내용 – AROv3용 최초 버전

이전 버전의 저자들

Oren Kashi <okashi@redhat.com>

Red Hat 기술 제품 마케팅 수석 시니어 매니저

Brooke Jackson, Nermina Miller, Jose Moreno, Ahmed Sabbour, Aditya Datar, Vince Power,
Alex Patterson, 그리고 이 가이드를 검토하면서 자신의 시간과 피드백을 아끼지 않는 분들께 감사드립니다.

12장

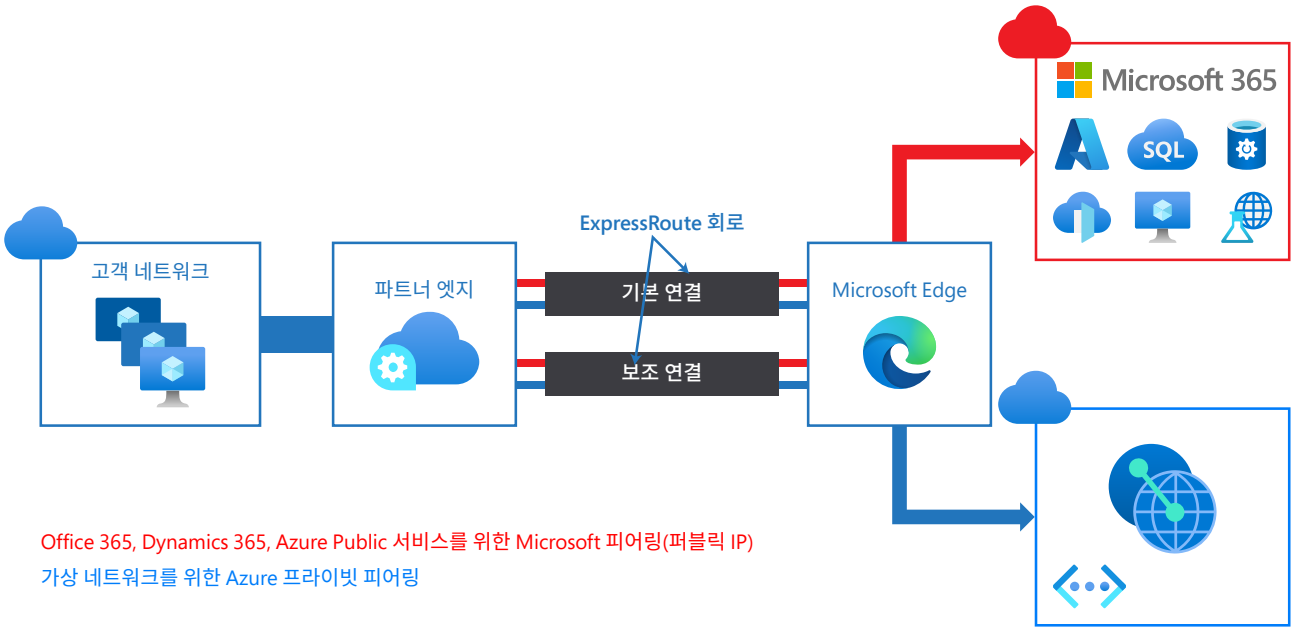
용어집

이 용어집은 이 가이드와 Azure Red Hat OpenShift 에코시스템에서 사용되는 일부 용어에 대한 유용한 빠른 참조를 제공합니다. 표제는 알파벳순입니다.

Azure ExpressRoute

ExpressRoute를 이용하면 연결성 제공업체의 도움을 받아 프라이빗 연결을 통해 온프레미스 네트워크를 Microsoft 클라우드로 확장할 수 있습니다. ExpressRoute를 통해 Microsoft Azure, Microsoft 365와 같은 Microsoft 클라우드 서비스에 대한 연결을 설정할 수 있습니다.

다음은 Microsoft Azure 도큐멘테이션에서 가져온 ExpressRoute 네트워크 다이어그램입니다.



ExpressRoute에 대해 자세히 알아보려면 Microsoft Azure 도큐멘테이션 페이지의 ExpressRoute란?을 참조하세요.

ExpressRoute가 Azure와 연동되는 방식을 이해하려면 *4장: 프로비저닝 전 – 엔터프라이즈 아키텍처에 관한 질문*을 참조하세요.

Azure 랜딩 구역

Azure 랜딩 구역은 확장, 보안 거버넌스, 네트워킹, 아이덴티티를 고려한 배포 패턴으로, Azure를 사용한 대규모 배포를 계획 중인 조직이 널리 사용하고 있습니다.

Azure 랜딩 구역 소개

현재 이 문서를 작성하는 시점에 개발 중인 커뮤니티 GitHub 프로젝트가 있습니다. 이 프로젝트는 Azure Red Hat OpenShift를 Azure 랜딩 구역 아키텍처로 배포하는 방법에 관해 몇 가지 사항을 권장합니다 (https://github.com/Azure/Enterprise-Scale/tree/main/workloads/ARO)

ReplicaSets

ReplicationController와 마찬가지로 ReplicaSet는 지정된 수의 포드 복제본이 언제든지 실행되도록 보장합니다. ReplicaSet와 ReplicationController의 차이점은 ReplicaSet가 세트 기반 선택기 요구 사항을 지원하는 반면, ReplicationController는 동등성 기반 선택기 요구 사항만 지원한다는 것입니다.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
    name: frontend-1
    labels:
        tier: frontend
spec:
    replicas: 3
    selector:
        matchLabels:
            tier: frontend
        matchExpressions:
        - {key: tier, operator: In, values: [frontend]}
    template:
        metadata:
            labels:
                tier: frontend
        spec:
            containers:
                - image: openshift/hello-openshift
                  name : helloworld
                  ports:
                  - containerPort: 8080
                    protocol: TCP
            restartPolicy: Always
```

위 코드에서 다음 사항을 확인할 수 있습니다.

- 리소스 세트에 대한 레이블 쿼리. matchLabels와 matchExpressions의 결과는 논리적으로 결합됩니다.
- 선택기와 일치하는 레이블로 리소스를 지정하는 동등성 기반 선택기
- 키를 필터링하는 세트 기반 선택기 이 선택기는 tier와 같은 키 및 frontend와 같은 값을 가진 모든 리소스를 선택합니다.

ReplicationController

ReplicationController는 지정된 수의 포드 복제본이 항상 실행되도록 보장합니다. 포드가 종료되거나 삭제되면 ReplicationController는 정의된 수만큼 최대한 더 많은 인스턴스화 작업을 수행합니다. 이와 마찬가지로 원하는 복제본보다 더 많이 실행되고 있으면 정의된 수량에 도달하는 데 필요한 만큼 삭제합니다.

ReplicationController 구성은 다음 항목으로 이루어져 있습니다.

- 원하는 복제본 개수(런타임에 수정할 수 있음)
- 복제된 포드를 생성할 때 사용할 포드 정의
- 관리형 포드 식별을 위한 선택기

- 선택기는 ReplicationController가 관리하는 포드에 할당된 레이블 세트입니다. 이 레이블은 ReplicationController가 인스턴스화하는 포드 정의에 포함되어 있습니다. ReplicationController는 필요에 따라 조정하기 위해 몇 개의 포드 인스턴스가 이미 실행 중인지 파악하기 위해 선택기를 사용합니다.

ReplicationController는 추적하지 않기 때문에 부하 또는 트래픽에 근거해 자동 확장을 수행하지 않습니다. 이 작업을 위해서는 복제본 개수를 외부 자동 스케일러가 조정할 수 있어야 합니다.

ReplicationController는 ReplicationController라고 하는 핵심 쿠버네티스 오브젝트입니다. 다음은 ReplicationController 정의의 예시입니다.

```
apiVersion: v1
kind: ReplicationController
metadata:
    name: frontend-1
spec:
    replicas: 1
    selector:
        name: frontend
    template:
        metadata:
            labels:
                name: frontend
        spec:
            containers:
                - image: openshift/hello-openshift
                  name: helloworld
                  ports:
                - containerPort: 8080
                  protocol: TCP
            restartPolicy: Always
```

위 코드에서 다음 사항을 확인할 수 있습니다.

- 실행할 포드의 복사본 개수
- 실행할 포드의 레이블 선택기
- 컨트롤러가 생성하는 포드용 템플릿
- 포드의 레이블은 레이블 선택기의 레이블을 포함해야 합니다.
- 매개 변수를 확장한 후 name을 구성하는 문자의 최대 길이는 63자입니다.

S2I(Source-to-Image)

S2I는 소스 코드에서 재현 가능한 컨테이너 이미지를 빌드하기 위한 툴킷이자 워크플로우입니다. S2I는 컨테이너 이미지에 소스 코드를 주입하고 컨테이너가 실행을 위해 해당 소스 코드를 준비하도록 하여 즉시 실행 가능한 이미지를 생성합니다. 자체 조립 빌더 이미지를 생성하면 컨테이너 이미지를 사용하여 런타임 환경의 버전을 관리하는 것과 똑같은 방식으로 빌드 환경을 버전 관리하고 제어할 수 있습니다.

Ruby와 같은 동적 언어의 경우 빌드 타임 환경과 런타임 환경은 일반적으로 동일합니다. S2I는 이 환경을 설명하는 빌더 이미지부터 시작하여 설치된 Ruby 애플리케이션을 설정하고 실행하는 데 필요한 Ruby, Bundler, Rake, Apache, GCC, 기타 패키지를 이용해 다음과 같은 단계를 수행합니다.

1. 알려진 디렉터리에 주입된 애플리케이션 소스로 빌더 이미지에서 컨테이너를 시작합니다.
2. 컨테이너 프로세스는 이 소스 코드를 적절한 실행 가능 설정으로 변환하는데, 이 경우에는 Bundler를 사용하여 종속성을 설치하고 Apache가 Ruby `config.ru` 파일을 찾도록 사전 구성된 디렉터리로 소스 코드를 이동하는 방식을 사용합니다.
3. 새로운 컨테이너를 커밋하고 이미지 엔트리 포인트를, Apache를 시작해 Ruby 애플리케이션을 호스팅할 스크립트(빌더 이미지에서 제공함)로 설정합니다.

C, Go 또는 Java와 같이 컴파일된 언어의 경우 컴파일에 필요한 종속성이 실제 런타임 아티팩트의 크기보다 훨씬 더 클 수 있습니다. 런타임 이미지를 간소하게 유지하기 위해 S2I는 다단계 빌드 프로세스를 지원합니다. 이 프로세스에서는 실행 파일 또는 Java WAR 파일과 같은 바이너리 아티팩트가 첫 번째 빌더 이미지에서 생성되고 추출된 후, 실행을 위한 올바른 위치에 실행 파일을 배치하는 기능만 수행하는 두 번째 런타임 이미지로 주입됩니다.

예를 들어 Tomcat(많이 사용되는 Java 웹 서버) 및 Maven을 위해 재현 가능한 빌드 파이프라인을 생성하려면 다음 단계를 수행합니다.

1. WAR 파일이 주입될 것으로 예상하는 OpenJDK 및 Tomcat 포함 빌더 이미지를 생성합니다.
2. 첫 번째 이미지 위에 Maven과 기타 모든 표준 종속성을 계층화하고 Maven 프로젝트가 주입될 것으로 예상하는 두 번째 이미지를 생성합니다.
3. Java 애플리케이션 소스와 Maven 이미지를 사용해 S2I를 호출하여 원하는 애플리케이션 WAR를 생성합니다.
4. 이전 단계의 WAR 파일과 초기 Tomcat 이미지를 사용해 S2I를 두 번째 호출하여 런타임 이미지를 생성합니다.

이미지 내에 빌드 로직을 배치하고 이미지를 여러 단계로 결합하면 빌드 툴을 프로덕션으로 배포하지 않고도 빌드 환경(동일한 JDK, 동일한 Tomcat JAR)과 가까운 곳에 런타임 환경을 유지할 수 있습니다.

S2I를 빌드 전략으로 사용하는 경우의 목표와 장점은 다음과 같습니다.

- **재현 가능성**: 빌드 환경을 컨테이너 이미지 내에 캡슐화하고 호출자를 위한 단순 인터페이스(주입된 소스 코드)를 정의하여 빌드 환경을 엄격하게 버전 관리할 수 있습니다. 재현 가능한 빌드는 컨테이너화된 인프라 내에서 보안 업데이트 및 지속적인 통합을 지원하기 위한 핵심 요구 사항이며, 빌더 이미지는 반복 가능성뿐 아니라 런타임 교체 능력도 보장하는 데 도움이 됩니다.
- **유연성**: Linux에서 실행할 수 있는 모든 기존 빌드 시스템은 컨테이너 내에서 실행할 수 있으며, 각 빌더는 더 큰 파이프라인의 일부일 수도 있습니다. 또한 애플리케이션 소스 코드를 처리하는 스크립트를 빌더 이미지에 주입할 수 있습니다. 이를 통해 작성자는 기존 이미지를 수정하여 소스 처리를 지원할 수 있습니다.
- **속도**: S2I는 단일 Dockerfile에 여러 계층을 빌드하는 대신 작성자가 단일 이미지 계층에 애플리케이션을 나타내도록 권장합니다. 따라서 생성 및 배포 시간이 단축되고 최종 이미지 출력을 더 효율적으로 제어할 수 있습니다.
- **보안**: Dockerfile을 사용한 빌드는 컨테이너에 대한 다수의 일반적인 운영 제어 기능 없이 실행되며, 일반적으로 루트로 실행되어 컨테이너 네트워크에 대한 액세스 권한을 보유하고 있습니다. 빌드는 하나의 컨테이너에서 시작되므로 S2I를 사용하여 빌더 이미지에 사용할 수 있는 권한을 제어할 수 있습니다. S2I는 OpenShift와 같은 플랫폼과 함께 관리자가 빌드 시간에 개발자가 갖는 권한을 엄격하게 제어하도록 지원할 수 있습니다.

경로 및 인그레스

Azure Red Hat OpenShift는 경로와 인그레스를 모두 지원합니다. 둘 다 www.example.com과 같은 DNS 이름을 사용해 서비스를 노출하는 데 사용됩니다. 따라서 외부 클라이언트는 서비스에 도달할 수 있습니다.

경로는 원래 Red Hat OpenShift 버전 3에서 구상되었습니다. 이 버전을 출시한 후 Red Hat은 쿠버네티스 커뮤니티와 협력해 현재 **인그레스**라는 오브젝트 API에서 이 기능을 표준화했습니다.

OpenShift Container Platform의 쿠버네티스 인그레스는 클러스터 내부에서 포드로 실행되는 공유 라우터 서비스로 인그레스 컨트롤러를 구현합니다. 인그레스 트래픽을 관리하는 가장 흔한 방법은 인그레스 컨트롤러를 이용하는 것입니다. 다른 정규 포드처럼 이 포드를 확장하고 복제할 수 있습니다. 이 라우터 서비스는 오픈소스 로드 밸런서 솔루션인 HAProxy에 기반을 두고 있습니다.

OpenShift Container Platform 경로는 클러스터의 서비스에 인그레스 트래픽을 제공합니다. 경로는 TLS 재암호화, TLS 패스스루, 블루-그린 배포를 위한 분할 트래픽 등 표준 쿠버네티스 인그레스 컨트롤러의 지원을 받지 않을 수 있는 고급 기능을 제공합니다.

인그레스 트래픽은 경로를 통해 클러스터의 서비스에 액세스합니다. 경로 및 인그레스는 인그레스 트래픽을 처리하기 위한 주요 리소스입니다. 인그레스는 외부 요청을 받아들여 경로에 따라 위임하는 등 경로와 유사한 기능들을 제공합니다. 하지만 인그레스를 통해 HTTP/2, HTTPS 및 **서버 이름 식별(SNI)**, 인증서가 있는 TLS와 같은 특정 유형의 연결만 허용할 수 있습니다. OpenShift Container Platform에서는 인그레스 리소스가 지정한 조건을 충족하기 위해 경로가 생성됩니다.

배포 및 배포 구성

복제 컨트롤러에서 빌드하는 Azure Red Hat OpenShift는 배포 개념을 이용해 소프트웨어 개발 및 배포 라이프사이클에 대한 확장된 지원을 추가합니다. 가장 간단한 경우 배포는 새로운 ReplicationController를 생성하여 포드를 시작할 수 있게 합니다. 하지만 Azure Red Hat OpenShift 배포는 기존 이미지 배포에서 새로운 배포로 전환할 수 있는 기능도 제공하며 ReplicationController를 생성하기 전과 후에 실행할 후크를 정의하기도 합니다.

Azure Red Hat OpenShift DeploymentConfig 오브젝트는 배포에 관한 세부 사항을 다음과 같이 정의합니다.

5. ReplicationController 정의의 여러 요소
6. 새로운 배포를 자동으로 생성하는 트리거
7. 배포 간 전환을 위한 전략
8. 라이프사이클 후크

배포가 트리거될 때마다 배포자 포드는 수동 또는 자동으로 배포를 관리합니다(기존 ReplicationController 축소, 새로운 ReplicationController 확장, 후크 실행 포함). 배포 포드는 배포 로그를 유지하기 위해 배포 완료 후 무기한 남아 있습니다. 배포가 다른 배포로 대체되면 필요한 경우 쉽게 롤백할 수 있도록 이전 ReplicationController가 유지됩니다.

배포를 생성하고 배포와 상호 작용하는 방법에 관한 자세한 내용은 배포 및 DeploymentConfigs를 참조하세요.

빌드 및 이미지 스트림

빌드는 입력 매개 변수를 오브젝트로 변환하는 프로세스입니다. 대부분의 경우 이 프로세스는 입력 매개 변수 또는 소스 코드를 실행 가능한 이미지로 변환하는 데 사용됩니다. BuildConfig 오브젝트는 전체 빌드 프로세스에 대한 정의입니다.

Azure Red Hat OpenShift는 빌드 이미지에서 Docker 형식 컨테이너를 생성하여 컨테이너 이미지 레지스트리로 푸시하는 방식으로 쿠버네티스를 사용합니다.

빌드 오브젝트는 빌드에 대한 입력, 빌드 프로세스를 완료할 필요성, 빌드 프로세스 로깅, 성공한 빌드에서 리소스 게시, 빌드의 최종 상태 게시와 같은 일반적인 특성을 공유합니다. 빌드는 리소스 제한을 이용해 CPU 사용량, 메모리 사용량, 빌드 또는 포드 실행 시간과 같은 제한 사항을 리소스에 지정합니다.

Azure Red Hat OpenShift 빌드 시스템은 빌드 API에 지정된 선택 가능 유형을 기반으로 하는 빌드 전략에 확장 가능한 지원을 제공합니다. 사용 가능한 세 가지 주요 빌드 전략은 다음과 같습니다.

- **Docker 빌드** – 소스 리포지토리에서 업로드 또는 풀링할 수 있는 Dockerfile 사용
- **S2I(Source-to-Image) 빌드** – 소스 리포지토리(예: Git)를 가져와 잘 알려진 언어 빌드 파일(예: Java 프로젝트용 Maven .pom 파일)에서 애플리케이션을 빌드하는 방법을 결정
- **사용자 정의 빌드**

기본적으로 Docker 빌드와 S2I 빌드가 지원됩니다.

결과 오브젝트는 빌드 생성에 사용되는 빌더에 따라 달라집니다. Docker 및 S2I 빌드의 경우, 결과 오브젝트는 실행 가능한 이미지입니다. 사용자 정의 빌드의 경우 결과 오브젝트는 빌더 이미지 작성자가 지정한 모든 오브젝트입니다.

작업

작업은 특정 이유로 포드를 생성하는 데 목적이 있다는 점에서 ReplicationController와 유사합니다. 차이점은 복제 컨트롤러가 지속적으로 실행될 포드를 위해 설계된 반면, 작업은 일회성 포드를 위해 설계되었다는 것입니다. 작업은 모든 성공적인 완료를 추적하고 지정된 양의 완료에 도달하면 작업도 완료됩니다.

작업 사용 방법에 관한 자세한 내용은 *작업* 토픽을 참조하세요.

컨테이너

Azure Red Hat OpenShift 애플리케이션의 기본 단위를 컨테이너라고 합니다. Linux 컨테이너 기술은 실행 중인 프로세스를 격리하기 위한 경량화 메커니즘이므로 지정된 리소스하고만 상호 작용할 수 있게 제한되어 있습니다.

여러 애플리케이션 인스턴스가 서로의 프로세스, 파일, 네트워크 등에 대한 가시성 없이 단일 호스트의 컨테이너에서 실행될 수 있습니다. 일반적으로 각 컨테이너는 임의의 워크로드에 사용될 수 있지만 웹 서버 또는 데이터베이스와 같은 단일 서비스(흔히 "마이크로서비스"라고 함)를 제공합니다.

컨테이너 레지스트리

Azure Red Hat OpenShift는 **OpenShift Container Registry(OCR)**라고 하는 통합 컨테이너 이미지 레지스트리를 제공합니다. 이 레지스트리는 새로운 이미지 리포지토리를 온디맨드 방식으로 자동 프로비저닝할 수 있는 기능을 추가로 제공합니다. 또한 애플리케이션 빌드가 결과 이미지를 푸시할 빌트인 위치를 사용자에게 제공합니다.

새 이미지가 OCR로 푸시될 때마다 레지스트리는 Azure Red Hat OpenShift에 새 이미지에 대해 알림으로써 네임스페이스, 이름, 이미지 메타데이터 등 새 이미지에 대한 모든 정보를 전달합니다. Azure Red Hat OpenShift를 구성하는 여러 요소는 새 이미지에 반응하여 새로운 빌드 및 배포를 생성합니다.

또한 Azure Red Hat OpenShift는 컨테이너 이미지 레지스트리 API를 Docker Hub, Azure Container Registry 등 이미지 소스로 구현하는 모든 서버를 활용합니다.

컨테이너 이미지

Azure Red Hat OpenShift 내 컨테이너는 Docker 형식의 컨테이너 이미지에 기반을 두고 있습니다. 이미지는 단일 컨테이너 실행을 위한 모든 요구 사항을 포함하는 바이너리일 뿐 아니라 단일 컨테이너 요구 사항 및 기능을 설명하는 메타데이터이기도 합니다.

이를 패키징 기술로 간주할 수 있습니다. 컨테이너 생성 시 컨테이너에 추가 액세스 권한을 부여하지 않는 한 컨테이너는 이미지에 정의된 리소스에만 액세스할 수 있습니다. 여러 호스트에 있는 여러 컨테이너에 동일 이미지를 배포하고 컨테이너 간에 부하를 분산함으로써 Azure Red Hat OpenShift는 이미지로 패키징된 서비스에 이중화 및 수평적 스케일링을 제공할 수 있습니다.

템플릿

템플릿은 Azure Red Hat OpenShift에서 만들 오브젝트 목록을 생성하기 위해 매개 변수화 및 처리할 수 있는 오브젝트 세트에 대해 설명합니다. 템플릿을 처리하여 사용자가 프로젝트 내에서 생성할 권한이 있는 모든 것(예: 서비스, 빌드 구성, 배포 구성)을 만들 수 있습니다. 템플릿은 템플릿에 정의된 모든 오브젝트에 적용할 레이블 세트를 정의할 수도 있습니다.

CLI를 사용하여 템플릿에서 오브젝트 목록을 생성할 수 있습니다. 템플릿이 프로젝트 또는 전역 템플릿 라이브러리로 업로드된 경우에는 웹 콘솔을 사용하여 오브젝트 목록을 생성할 수 있습니다. 큐레이팅된 템플릿 세트인 경우 OpenShift 이미지 스트림 및 템플릿 라이브러리를 확인하세요.

프드 및 서비스

Azure Red Hat OpenShift는 호스트 하나에 함께 배포되는 한 개 이상의 컨테이너이자 정의, 배포, 관리할 수 있는 최소 컴퓨팅 단위인 포드라는 쿠버네티스 개념을 사용합니다.

포드는 컨테이너에 대한 머신 인스턴스(물리 또는 가상)와 대략적으로 동일합니다. 각 포드는 자체 내부 IP 주소를 할당받으므로 전체 포트 공간을 소유하며, 포드 내 컨테이너는 자체 로컬 스토리지 및 네트워킹을 공유할 수 있습니다.

포드에는 라이프사이클이 있어 포드가 정의된 후 노드에서 실행하도록 할당된 다음 컨테이너가 종료될 때까지 실행되거나 다른 이유로 제거됩니다. 포드는 정책 및 종료 코드에 따라 종료 후 제거되거나 자체 컨테이너의 로그에 대한 액세스를 지원하기 위해 유지될 수 있습니다.

Azure Red Hat OpenShift는 포드를 대체로 변경할 수 없는 것으로 취급하므로 포드가 실행 중일 때는 포드 정의를 변경할 수 없습니다. Azure Red Hat OpenShift는 기존 포드를 종료하고 구성, 기본 이미지 또는 둘 다 수정하여 다시 생성함으로써 변경 사항을 구현합니다. 포드의 런타임 구성 요소는 소모품으로 취급되므로 정의된 컨테이너 이미지에서 다시 생성됩니다. 따라서 포드는 대체로 사용자가 직접 관리하는 대신 더 높은 수준의 컨트롤러로 관리해야 합니다.

프로젝트 및 사용자

프로젝트는 추가 주석이 있는 쿠버네티스 네임스페이스이며, 일반 사용자의 리소스 액세스 권한을 관리하는 중심 수단입니다. 사용자 커뮤니티는 프로젝트를 통해 다른 커뮤니티와 별도로 콘텐츠를 구성하고 관리할 수 있습니다. 사용자는 관리자에게서 프로젝트에 액세스할 수 있는 권한을 받아야 합니다. 또는 프로젝트를 생성하도록 허용된 경우 자신의 프로젝트에 자동으로 액세스할 수 있어야 합니다. 프로젝트에는 별도의 이름, `displayName`, description이 있을 수 있습니다.

필수 사항인 name은 고유 식별자로서 CLI 툴 또는 API를 사용할 때 가장 가시성이 높습니다. name을 구성하는 문자의 최대 길이는 63자입니다. 선택 사항인 `displayName`은 프로젝트가 웹 콘솔에서 표시되는 방식입니다(`name`으로 기본 설정되어 있음). 선택 사항인 description은 프로젝트에 대한 더 자세한 설명일 수 있으며 웹 콘솔에서도 볼 수 있습니다.

개발자와 관리자는 CLI 또는 웹 콘솔을 사용하여 프로젝트와 상호 작용할 수 있습니다.