OpenShift Container Platform 4.6

보안 및 컴플라이언스

OpenShift Container Platform의 보안 학습 및 관리

Last Updated: 2023-04-07

OpenShift Container Platform 4.6 보안 및 컴플라이언스

OpenShift Container Platform의 보안 학습 및 관리

법적 공지

초록

이 문서에서는 컨테이너 보안, 인증서 구성 및 암호화를 사용하여 클러스터를 보호하는 방법을 설명합니다.

차례

1장. OPENSHIFT CONTAINER PLATFORM 보안 및 컴플라이언스

1.1. 보안 개요

OpenShift Container Platform 클러스터의 다양한 측면을 적절하게 보호하는 방법을 이해하는 것이 중요합니다.

컨테이너 보안
OpenShift Container Platform 보안을 이해하기 위한 좋은 시작점은 컨테이너 보안 이해의 개념을 검토하는 것입니다. 이 섹션에서는 호스트 계층, 컨테이너 및 오케스트레이션 계층, 빌드 및 애플리케이션 계층에 대한 솔루션을 포함하여 OpenShift Container Platform에서 사용할 수 있는 컨테이너 보안 조치의 수준 높은 검토 단계를 제공합니다. 이 섹션에는 다음 주제에 대한 정보도 포함되어 있습니다.

- 컨테이너 보안이 중요한 이유 및 기존 보안 표준과의 비교
- 호스트(RHCOS 및 RHEL) 계층에서 제공하는 컨테이너 보안 조치와 OpenShift Container Platform에서 제공하는 컨테이너 보안 조치
- 컨테이너 콘텐츠 및 취약점의 소스를 평가하는 방법
- 컨테이너 콘텐츠를 사전 예방식으로 확인하기 위해 빌드 및 배포 프로세스를 디자인하는 방법
- 인증 및 권한 부여를 통해 컨테이너에 대한 액세스를 제어하는 방법
- OpenShift Container Platform에서 네트워킹 및 연결된 스토리지를 보호하는 방법
- API 관리 및 SSO에 사용하는 컨테이너화된 솔루션

감사
OpenShift Container Platform 감사에서는 시스템의 개별 사용자, 관리자 또는 기타 구성 요소가 시스템에 영향을 준 활동 시퀀스를 설명하는 보안 관련 레코드 집합을 제공합니다. 관리자는 감사 로그 정책 구성 및 감사 로그 보기를 수행할 수 있습니다.

인증서
인증서는 다양한 구성 요소에서 클러스터에 대한 액세스 권한을 검증하는 데 사용됩니다. 관리자는 기본 ingress 인증서를 교체하거나 API 서버 인증서를 추가하거나 서비스 인증서를 추가할 수 있습니다.

클러스터에서 사용하는 인증서 유형에 대한 세부 정보를 검토할 수도 있습니다.

- API 서버의 사용자 제공 인증서
- 프록시 인증서
- 서비스 CA 인증서
- 노드 인증서
- 부트스트랩 인증서
- etcd 인증서
- OLM 인증서
- 기본 수신을 위한 사용자 제공 인증서
- 수신 인증서

- 모니터링 및 클러스터 로깅 Operator 구성요소 인증서
- 컨트롤 플레인 인증서

데이터 암호화
클러스터에 etcd 암호화를 활성화 하여 추가 데이터 보안 계층을 제공할 수 있습니다. 예를 들어 etcd 백업이 잘못된 당사자에게 노출되는 경우 중요한 데이터의 손실을 방지할 수 있습니다.

취약점 검사
관리자는 Red Hat Quay Container Security Operator를 사용하여 취약점 검사를 실행하고 탐지된 취약점에 대한 정보를 검토할 수 있습니다.

1.2. 컴플라이언스 개요

많은 OpenShift Container Platform 고객은 시스템을 프로덕션 환경에 도입하기 전에 일정 수준의 규제 준비 또는 컴플라이언스를 갖춰야 합니다. 이러한 규제 준비는 국가 표준, 산업 표준 또는 조직의 기업 거버넌스 프레임워크에 의해 부과될 수 있습니다.

컴플라이언스 확인
관리자는 Compliance Operator를 사용하여 규정 준수 검사를 실행하고 발견된 문제에 대한 수정을 권장할 수 있습니다.

파일 무결성 검사
관리자는 File Integrity Operator를 사용하여 클러스터 노드에서 파일 무결성 검사를 지속적으로 실행하고 수정된 파일의 로그를 제공할 수 있습니다.

1.3. 추가 리소스

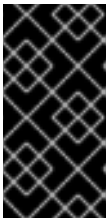
- 인증 이해
- 내부 OAuth 서버 구성
- ID 공급자 구성 이해
- RBAC를 사용하여 권한 정의 및 적용
- 보안 컨텍스트 제약 조건 관리

2장. 컨테이너 보안

2.1. 컨테이너 보안 이해

컨테이너화된 애플리케이션은 여러 수준의 보안에 의존합니다.

- 컨테이너 보안은 신뢰할 수 있는 기본 컨테이너 이미지로 시작하며 CI/CD 파이프라인에 따라 진행하면서 컨테이너 빌드 프로세스를 계속합니다.



중요

기본적으로 이미지 스트림은 자동으로 업데이트되지 않습니다. 이 기본 동작은 이미지 스트림에서 참조하는 이미지에 대한 보안 업데이트가 자동으로 발생하지 않기 때문에 보안 문제가 발생할 수 있습니다. 이 기본 동작을 재정의하는 방법에 대한 자세한 내용은 주기적으로 imagestreamtag 가져오기 구성 을 참조하십시오.

- 배포된 컨테이너의 보안은 컨테이너가 보안 운영 체제 및 네트워크에서 실행 중이며 컨테이너와 상호 작용하는 사용자 및 호스트와 컨테이너 자체 사이에 확실한 경계가 설정되어 있는지에 따라 결정됩니다.

- 컨테이너 이미지에서 취약점을 스캔하고 취약한 이미지를 효율적으로 수정 및 교체하는 기능이 있어야 지속적인 보안이 가능합니다.

OpenShift Container Platform과 같은 플랫폼이 기본적으로 제공하는 보안 외에도 조직에는 자체 보안 요구 사항이 있습니다. OpenShift Container Platform을 데이터 센터로 가져오려면 일정 수준의 컴플라이언스 확인이 필요할 수 있습니다.

마찬가지로 조직의 보안 표준을 충족하기 위해 자체 에이전트, 특수 하드웨어 드라이버 또는 암호화 기능을 OpenShift Container Platform에 추가해야 할 수도 있습니다.

이 가이드에서는 호스트 계층, 컨테이너 및 오케스트레이션 계층, 빌드 및 애플리케이션 계층의 솔루션을 포함하여 OpenShift Container Platform에서 사용 가능한 컨테이너 보안 조치의 수준 높은 검토 단계를 제공합니다. 그런 다음 보안 조치를 달성하는 데 도움이 되는 특정 OpenShift Container Platform 설명서를 표시합니다.

이 안내서에는 다음 정보가 포함되어 있습니다.

- 컨테이너 보안이 중요한 이유 및 기존 보안 표준과의 비교

- 호스트(RHCOS 및 RHEL) 계층에서 제공하는 컨테이너 보안 조치와 OpenShift Container Platform에서 제공하는 컨테이너 보안 조치

- 컨테이너 콘텐츠 및 취약점의 소스를 평가하는 방법

- 컨테이너 콘텐츠를 사전 예방식으로 확인하기 위해 빌드 및 배포 프로세스를 디자인하는 방법

- 인증 및 권한 부여를 통해 컨테이너에 대한 액세스를 제어하는 방법

- OpenShift Container Platform에서 네트워킹 및 연결된 스토리지를 보호하는 방법

- API 관리 및 SSO에 사용하는 컨테이너화된 솔루션

이 가이드의 목표는 컨테이너화된 워크로드에 OpenShift Container Platform을 사용하여 얻을 수 있는 뛰어난 보안 이점과 Red Hat 에코시스템을 통해 컨테이너를 안전하게 만들고 유지하는 방법을 이해하는 것입니다. 또한 조직의 보안 목표를 달성하기 위해 OpenShift Container Platform에 참여하는 방법을 이해

하는 데 도움이 됩니다.

2.1.1. 컨테이너 정의

컨테이너에서는 애플리케이션과 모든 종속 항목을 변경하지 않고 개발에서 테스트 및 프로덕션으로 승격할 수 있는 단일 이미지로 패키징합니다. 컨테이너는 다른 컨테이너와 밀접하게 작동하는 더 큰 애플리케이션의 일부일 수 있습니다.

컨테이너에서는 물리 서버, 가상 머신(VM) 및 프라이빗 또는 퍼블릭 클라우드와 같은 환경 및 여러 배치 대상 사이에 일관성을 제공합니다.

컨테이너를 사용하면 다음과 같은 이점이 있습니다.

인프라	애플리케이션
공유 Linux 운영 체제 커널의 샌드박스 애플리케이션 프로세스	내 애플리케이션 및 모든 종속 항목 패키징
가상 머신보다 단순하고 가벼우며 밀도가 높음	몇 초 안에 모든 환경에 배포하고 CI/CD 활성화
다양한 환경에 이식 가능	컨테이너화된 구성요소에 쉽게 액세스 및 공유

Linux 컨테이너에 관한 자세한 내용은 Red Hat Customer Portal의 Linux 컨테이너 이해 를 참조하십시오. RHEL 컨테이너 툴에 관해 알아보려면 RHEL 제품 설명서의 컨테이너 빌드, 실행 및 관리 를 참조하십시오.

2.1.2. OpenShift Container Platform의 정의

컨테이너화된 애플리케이션이 배포, 실행 및 관리되는 방식을 자동화하는 것이 OpenShift Container Platform과 같은 플랫폼의 역할입니다. OpenShift Container Platform의 핵심은 쿠버네티스 프로젝트를 사용하여 확장 가능한 데이터 센터의 여러 노드에서 컨테이너를 조정하는 엔진을 제공하는 것입니다.

쿠버네티스는 프로젝트에서 지원 가능성을 보장하지 않는 여러 다른 운영 체제 및 애드온 구성요소를 사용하여 실행할 수 있는 프로젝트입니다. 결과적으로 쿠버네티스 플랫폼마다 보안이 다를 수 있습니다.

OpenShift Container Platform은 쿠버네티스 보안을 잠그고 다양한 확장 구성요소와 플랫폼을 통합하도록 설계되었습니다. 이를 위해 OpenShift Container Platform에서는 운영 체제, 인증, 스토리지, 네트워킹, 개발 툴, 기본 컨테이너 이미지 및 기타 여러 구성요소를 포함하는 광범위한 오픈소스 기술의 Red Hat 에코 시스템을 활용합니다.

OpenShift Container Platform은 플랫폼에서 실행 중인 컨테이너화된 애플리케이션 외에도 플랫폼 자체의 취약점을 발견하고 신속하게 수정 사항을 배포하는 Red Hat의 환경을 활용할 수 있습니다. Red Hat 환경에서는 새로운 구성요소가 사용 가능하게 되면 OpenShift Container Platform과 효율적으로 통합하고 개별 고객 요구 사항에 맞게 기술을 조정하는 범위까지 기능이 확장됩니다.

추가 리소스

- OpenShift Container Platform 아키텍처
- OpenShift 보안 가이드

2.2. 호스트 및 VM 보안 이해

컨테이너와 가상 머신 모두 호스트에서 실행 중인 애플리케이션을 운영 체제 자체에서 분리하는 방법을 제공합니다. OpenShift Container Platform에서 사용하는 운영 체제인 RHCOS를 이해하면 호스트 시스템이 컨테이너와 호스트를 서로 보호하는 방법을 알 수 있습니다.

2.2.1. RHCOS(Red Hat Enterprise Linux CoreOS)의 컨테이너 보안

컨테이너를 사용하면 동일한 커널 및 컨테이너 런타임을 통해 각 컨테이너를 가동하여 동일한 호스트에서 실행되도록 많은 애플리케이션을 배포하는 작업이 단순화됩니다. 애플리케이션은 많은 사용자가 소유할 수 있으며, 개별적으로 유지되기 때문에 서로 다르거나 호환되지 않는 버전의 애플리케이션도 문제없이 동시에 실행될 수 있습니다.

Linux에서 컨테이너는 특별한 유형의 프로세스이므로 컨테이너 보안은 여러 가지 면에서 기타 실행 중인 프로세스를 보호하는 것과 비슷합니다. 컨테이너를 실행하는 환경은 컨테이너를 서로 보호할 뿐만 아니라 호스트에서 실행 중인 다른 프로세스 및 컨테이너로부터 호스트 커널을 보호할 수 있는 운영 체제로 시작합니다.

OpenShift Container Platform 4.6은 RHEL(Red Hat Enterprise Linux)을 작업자 노드로 사용하는 옵션과 함께 RHCOS 호스트에서 실행되므로 다음 개념이 배포된 모든 OpenShift Container Platform 클러스터에 기본적으로 적용됩니다. 이러한 RHEL 보안 기능은 OpenShift에서 컨테이너를 더 안전하게 실행할 수 있게 하는 핵심 요소입니다.

- *Linux 네임스페이스*를 사용하면 특정 글로벌 시스템 리소스를 추상화하여 네임스페이스에서 처리할 별도의 인스턴스로 표시할 수 있습니다. 따라서 여러 컨테이너가 충돌없이 동일한 컴퓨팅 리소스를 동시에 사용할 수 있습니다. 기본적으로 호스트와 분리된 컨테이너 네임스페이스에는 마운트 테이블, 프로세스 테이블, 네트워크 인터페이스, 사용자, 제어 그룹, UTS 및 IPC 네임스페이스가 있습니다. 호스트 네임스페이스에 직접 액세스해야 하는 컨테이너에는 높은 권한이 있어야 해당 액세스를 요청할 수 있습니다. 네임스페이스 유형에 관한 자세한 내용을 포함하는 RHEL 7 컨테이너 설명서는 Red Hat Systems의 컨테이너 개요 를 참조하십시오.

- *SELinux*에서는 컨테이너 간에 서로 격리하고 컨테이너를 호스트에서 격리된 상태로 유지하는 추가 보안 계층을 제공합니다. 관리자는 SELinux를 사용하여 모든 사용자, 애플리케이션, 프로세스 및 파일에 MAC(필수 액세스 제어)를 적용할 수 있습니다.

주의

RHCOS 노드에서 SELinux를 비활성화하는 것은 지원되지 않습니다.

- *CGroup*(제어 그룹)은 프로세스 컬렉션의 리소스 사용량(CPU, 메모리, 디스크 I/O, 네트워크 등)을 제한, 설명 및 격리합니다. CGroup을 사용하면 동일한 호스트의 컨테이너가 서로 영향을 주지 않습니다.

- *보안 컴퓨팅 모드(seccomp)* 프로파일은 사용 가능한 시스템 호출을 제한하기 위해 컨테이너와 연관될 수 있습니다. seccomp에 관한 자세한 내용은 OpenShift 보안 안내서 의 94페이지를 참조하십시오.

- *RHCOS*를 사용하여 컨테이너를 배포하면 호스트 환경을 최소화하고 컨테이너에 맞게 조정하여 공격 면적을 줄입니다. CRI-O 컨테이너 엔진 은 데스크톱 지향 독립 실행형 기능을 구현하는 다른 컨테이너 엔진과 달리 컨테이너를 실행하고 관리하기 위해 쿠버네티스 및 OpenShift에 필요한 기능만 구현하여 공격 면적을 더 줄입니다.

RHCOS는 OpenShift Container Platform 클러스터에서 컨트롤 플레인(마스터) 및 작업자 노드로 작동하도록 특별히 구성된 RHEL(Red Hat Enterprise Linux) 버전입니다. 따라서 RHCOS는 쿠버네티스 및 OpenShift 서비스와 함께 컨테이너 워크로드를 효율적으로 실행하도록 조정됩니다.

OpenShift Container Platform 클러스터에서 RHCOS 시스템을 추가로 보호하려면 호스트 시스템 자체를 관리하거나 모니터링하는 컨테이너를 제외한 대부분의 컨테이너는 루트가 아닌 사용자로 실행해야 합니다. 고유한 OpenShift Container Platform 클러스터를 보호하는 데 권장되는 모범 사례는 권한 수준을 낮추거나 가능한 최소 권한으로 컨테이너를 생성하는 것입니다.

추가 리소스

- 노드에서 리소스 제약 조건을 적용하는 방법
- 보안 컨텍스트 제약 조건 관리
- 사용 가능한 플랫폼
- 사용자가 제공하는 인프라를 사용하는 경우 클러스터의 시스템 요구 사항
- RHCOS 구성 방법 선택
- Ignition
- 커널 인수
- 커널 모듈
- FIPS 암호화
- 디스크 암호화
- Chrony 타임 서비스
- OpenShift Container Platform 클러스터 업데이트

2.2.2. 가상화 및 컨테이너 비교

기존 가상화에서는 동일한 물리적 호스트에서 애플리케이션 환경을 분리한 상태로 유지할 수 있는 또 다른 방법을 제공합니다. 그러나 가상 머신은 컨테이너와 다른 방식으로 작동합니다. 가상화는 게스트 가상 머신(VM)을 가동시키는 하이퍼바이저를 사용하며, 각 가상 머신에는 실행 중인 커널로 표시되는 자체 운영 체제(OS)와 실행 중인 애플리케이션 및 해당 종속 항목이 있습니다.

VM을 사용하면 하이퍼바이저에서 게스트 간에 서로 분리하고 호스트 커널에서 게스트를 분리합니다. 하이퍼바이저에 액세스하는 개인과 프로세스 수가 적어지므로 물리 서버의 공격 면적을 줄입니다. 보안은 여전히 모니터링해야 하지만, 하나의 게스트 VM은 하이퍼바이저 버그를 사용하여 다른 VM 또는 호스트 커널에 액세스할 수 있습니다. 또한 OS에 패치가 필요한 경우 해당 OS를 사용하는 모든 게스트 VM에서 패치를 적용해야 합니다.

컨테이너는 게스트 VM 내부에서 실행될 수 있으며 이와 같은 조치 바람직한 경우가 있을 수 있습니다. 예를 들어, 애플리케이션을 클라우드로 수정 없이 그대로 리프트 앤 시프트(lift-and-shift) 방식으로 배포하고 이동하기 위해 컨테이너에 기존 애플리케이션을 배포할 수 있습니다.

그러나 단일 호스트에서 컨테이너를 분리하면 더 가볍고 유연하며 쉽게 확장되는 배포 솔루션이 제공됩니다. 이 배포 모델은 특히 클라우드 네이티브 애플리케이션에 적합합니다. 컨테이너는 일반적으로 VM보다 훨씬 작으며 메모리와 CPU 사용량이 적습니다.

컨테이너와 VM의 차이점에 관해 알아보려면 RHEL 7 컨테이너 설명서의 Linux 컨테이너와 KVM 가상화 비교를 참조하십시오.

2.2.3. OpenShift Container Platform 보호

OpenShift Container Platform을 배포할 때 설치 프로그램 프로비저닝 인프라(사용 가능한 플랫폼이 여러 개) 또는 사용자가 프로비저닝한 자체 인프라 중에서 선택할 수 있습니다. FIPS 컴플라이언스 활성화 또는 첫 부팅 시 필요한 커널 모듈 추가와 같은 일부 낮은 수준의 보안 관련 구성에 사용자가 프로비저닝한 인프라를 활용할 수 있습니다. 마찬가지로 사용자 프로비저닝 인프라는 연결 해제된 OpenShift Container Platform 배포에 적합합니다.

OpenShift Container Platform의 보안 향상 및 기타 구성 변경과 관련된 목표는 다음과 같습니다.

- 기본 노드를 최대한 일반적으로 유지합니다. 비슷한 노드를 신속하고 규범에 맞는 방식으로 쉽게 제거하고 구동할 수 있어야 합니다.
- 노드를 일회성으로 직접 변경하지 말고 최대한 OpenShift Container Platform에서 노드 수정을 관리합니다.

이러한 목표를 달성하기 위해 대부분의 노드 변경은 설치 중 Ignition을 사용하거나 나중에 Machine Config Operator가 노드 세트에 적용하는 MachineConfig를 사용하여 수행해야 합니다. 이러한 방식으로 수행할 수 있는 보안 관련 구성 변경의 예는 다음과 같습니다.

- 커널 인수 추가
- 커널 모듈 추가
- FIPS 암호화 지원 활성화
- 디스크 암호화 구성
- chrony 타임 서비스 구성

Machine Config Operator 외에도 CVO(Cluster Version Operator)에서 관리하며 OpenShift Container Platform 인프라를 구성하는 데 사용할 수 있는 Operator는 여러 가지가 있습니다. CVO를 사용하면 OpenShift Container Platform 클러스터 업데이트의 여러 요소를 자동화할 수 있습니다.

2.3. RHCOS 강화

RHCOS는 RHCOS 노드에 필요한 변경이 거의 없이 OpenShift Container Platform에 배포되도록 생성되고 조정되었습니다. OpenShift Container Platform을 채택한 모든 조직에는 시스템 강화를 위한 고유 요구 사항이 있습니다. OpenShift 고유 수정 사항 및 기능(예: 제한된 불변성을 제공하는 Ignition, ostree 및 읽기 전용 **/usr** 등)이 추가된 RHEL 시스템으로 RHCOS는 다른 RHEL 시스템과 마찬가지로 강화될 수 있습니다. 차이점은 강화 관리 방법에 있습니다.

OpenShift Container Platform과 쿠버네티스 엔진의 주요 기능은 필요에 따라 애플리케이션과 인프라를 빠르게 확장하고 축소할 수 있다는 것입니다. 불가피한 경우가 아니라면 호스트에 로그인하고 소프트웨어를 추가하거나 설정을 변경하여 RHCOS를 직접 변경하지 않게 합니다. OpenShift Container Platform 설치 프로그램 및 컨트롤 플레인에서 RHCOS의 변경 사항을 관리하여 수동 조작없이 새 노드를 구동할 수 있습니다.

따라서 보안 요구 사항을 충족하기 위해 OpenShift Container Platform에서 RHCOS 노드를 강화하려는 경우 강화할 대상과 강화를 수행하는 방법을 모두 고려해야 합니다.

2.3.1. RHCOS에서 강화할 대상 선택

RHEL 8 보안 강화 가이드에서는 RHEL 시스템의 보안 접근 방법을 설명합니다.

이 가이드를 사용하여 암호화에 접근하고 취약점을 평가하며 다양한 서비스에 대한 위협을 평가하는 방법을 알아봅니다. 마찬가지로 컴플라이언스 표준을 스캔하고, 파일 무결성을 확인하며, 감사를 수행하고, 스토리지 장치를 암호화하는 방법을 배울 수 있습니다.

강화하려는 기능에 관한 지식을 통해 RHCOS에서 강화할 방법을 결정할 수 있습니다.

2.3.2. RHCOS 강화 방법 선택

OpenShift Container Platform에서 RHCOS 시스템을 직접 수정하지 않는 것이 좋습니다. 대신 작업자 노드 및 컨트롤 플레인 노드 (마스터 노드라고도 함)와 같은 노드 풀에서 시스템을 수정하는 것을 고려해야 합니다. 베어 메탈이 아닌 설치에서 새 노드가 필요한 경우 원하는 유형의 새 노드를 요청할 수 있으며 RHCOS 이미지와 이전에 수정한 사항으로 노드를 생성합니다.

설치 전, 설치 중 및 클러스터가 가동되어 실행된 후에 RHCOS를 수정할 기회가 있습니다.

2.3.2.1. 설치 전 강화

베어 메탈 설치의 경우 OpenShift Container Platform 설치를 시작하기 전에 RHCOS에 강화 기능을 추가할 수 있습니다. 예를 들어 RHCOS 설치 프로그램을 부팅할 때 커널 옵션을 추가하여 SELinux 부울 또는 다양한 하위 수준 설정(예: 대칭 멀티스레딩)과 같은 보안 기능을 켜거나 끌 수 있습니다.

베어 메탈 RHCOS 설치는 더 어렵지만 OpenShift Container Platform 설치를 시작하기 전에 운영 체제를 변경할 수 있는 기회를 제공합니다. 디스크 암호화 또는 특수 네트워킹 설정과 같은 특정 기능을 가능한 빨리 설정해야 할 때 중요할 수 있습니다.

주의

RHCOS 노드에서 SELinux를 비활성화하는 것은 지원되지 않습니다.

2.3.2.2. 설치 중 강화

OpenShift 설치 프로세스를 중단하고 Ignition 구성을 변경할 수 있습니다. Ignition 구성을 통해 자체 파일 및 systemd 서비스를 RHCOS 노드에 추가할 수 있습니다. 또한 설치에 사용되는 **install-config.yaml** 파일에서 기본 보안 관련 사항을 변경할 수 있습니다. 이 방식으로 추가된 콘텐츠는 각 노드의 첫 부팅 시 사용할 수 있습니다.

2.3.2.3. 클러스터가 실행된 후 강화

OpenShift Container Platform 클러스터가 가동되어 실행된 후 RHCOS에 강화 기능을 적용하는 방법은 몇 가지가 있습니다.

- 데몬 세트: 모든 노드에서 서비스를 실행해야 하는 경우 Kubernetes **DaemonSet** 오브젝트로 해당 서비스를 추가할 수 있습니다.
- 머신 구성: **MachineConfig** 오브젝트에는 동일한 형식의 Ignition 구성 서브 세트가 포함되어 있습니다. 머신 구성을 모든 작업자 또는 컨트롤 플레인 노드에 적용하면 클러스터에 추가된 동일한 유형의 다음 노드에 동일한 변경 사항이 적용될 수 있습니다.

여기에 언급된 모든 기능은 OpenShift Container Platform 제품 설명서에 설명되어 있습니다.

추가 리소스

- OpenShift 보안 가이드
- RHCOS 구성 방법 선택
- 노드 수정
- 수동으로 설치 구성 파일 생성
- 쿠버네티스 매니페스트 및 Ignition 설정 파일 생성
- ISO 이미지를 사용하여 RHCOS(Red Hat Enterprise Linux CoreOS) 머신 생성
- 노드의 사용자 정의
- 노드에 커널 인수 추가
- 설치 구성 매개변수 - **fips** 참조
- FIPS 암호화 지원
- RHEL 핵심 암호화 구성요소

2.4. 컨테이너 이미지 서명

Red Hat은 Red Hat Container Registries에 있는 이미지에 대한 서명을 제공합니다. 이러한 서명은 MCO(Machine Config Operator)를 사용하여 OpenShift Container Platform 4 클러스터로 가져올 때 자동으로 확인할 수 있습니다.

Quay.io는 OpenShift Container Platform을 구성하는 대부분의 이미지를 제공하며 릴리스 이미지만 서명됩니다. 릴리스 이미지는 승인된 OpenShift Container Platform 이미지를 참조하여 공급 체인 공격에 대한 보안 수준을 제공합니다. 그러나 로깅, 모니터링 및 서비스 메시와 같은 OpenShift Container Platform에 대한 일부 확장 기능은 OLM(Operator Lifecycle Manager)에서 Operator로 제공됩니다. 이러한 이미지는 Red Hat Ecosystem Catalog 컨테이너 이미지 레지스트리에서 제공됩니다.

Red Hat 레지스트리와 인프라 간의 이미지의 무결성을 확인하려면 서명 확인을 활성화하십시오.

2.4.1. Red Hat Container Registries에 대한 서명 확인 활성화

컨테이너 서명 검증을 활성화하려면 레지스트리 URL을 sigstore에 연결한 다음 이미지를 확인하는 키를 지정하는 파일이 필요합니다.

프로세스

1. 레지스트리 URL을 sigstore에 연결하고 이미지를 확인하는 키를 지정하는 파일을 만듭니다.
   - **policy.json** 파일을 생성합니다.

```
$ cat > policy.json <<EOF
{
  "default": [
    {
      "type": "insecureAcceptAnything"
    }
  ],
```

```
  "transports": {
    "docker": {
      "registry.access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ],
      "registry.redhat.io": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ]
    },
    "docker-daemon": {
      "": [
        {
          "type": "insecureAcceptAnything"
        }
      ]
    }
  }
}
EOF
```

- **registry.access.redhat.com.yaml** 파일을 생성합니다.

```
$ cat <<EOF > registry.access.redhat.com.yaml
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
EOF
```

- **registry.redhat.io.yaml** 파일을 생성합니다.

```
$ cat <<EOF > registry.redhat.io.yaml
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
EOF
```

2. 머신 구성 템플릿에 사용할 **base64** 인코딩 형식으로 파일을 설정합니다.

```
$ export ARC_REG=$( cat registry.access.redhat.com.yaml | base64 -w0 )
$ export RIO_REG=$( cat registry.redhat.io.yaml | base64 -w0 )
$ export POLICY_CONFIG=$( cat policy.json | base64 -w0 )
```

3. 내보낸 파일을 작업자 노드의 디스크에 쓰는 머신 구성을 생성합니다.

```
$ cat > 51-worker-rh-registry-trust.yaml <<EOF
apiVersion: machineconfiguration.openshift.io/v1
```

```
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 51-worker-rh-registry-trust
spec:
  config:
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
      version: 2.2.0
    networkd: {}
    passwd: {}
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,${ARC_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.access.redhat.com.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${RIO_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.redhat.io.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${POLICY_CONFIG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/policy.json
  osImageURL: ""
EOF
```

4. 생성된 머신 구성을 적용합니다.

```
$ oc apply -f 51-worker-rh-registry-trust.yaml
```

5. 내보낸 파일을 마스터 노드의 디스크에 쓰는 머신 구성을 생성합니다.

```
$ cat > 51-master-rh-registry-trust.yaml <<EOF
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 51-master-rh-registry-trust
spec:
  config:
    ignition:
      config: {}
```

```
    security:
      tls: {}
    timeouts: {}
    version: 2.2.0
  networkd: {}
  passwd: {}
  storage:
    files:
    - contents:
        source: data:text/plain;charset=utf-8;base64,${ARC_REG}
        verification: {}
      filesystem: root
      mode: 420
      path: /etc/containers/registries.d/registry.access.redhat.com.yaml
    - contents:
        source: data:text/plain;charset=utf-8;base64,${RIO_REG}
        verification: {}
      filesystem: root
      mode: 420
      path: /etc/containers/registries.d/registry.redhat.io.yaml
    - contents:
        source: data:text/plain;charset=utf-8;base64,${POLICY_CONFIG}
        verification: {}
      filesystem: root
      mode: 420
      path: /etc/containers/policy.json
  osImageURL: ""
EOF
```

6. 마스터 머신 구성 변경 사항을 클러스터에 적용합니다.

```
$ oc apply -f 51-master-rh-registry-trust.yaml
```

2.4.2. 서명 확인 구성 확인

머신 구성을 클러스터에 적용한 후 머신 구성 컨트롤러에서 새 **MachineConfig** 오브젝트를 감지하고 새로운 **rendered-worker-<hash>** 버전을 생성합니다.

사전 요구 사항

- 머신 구성 파일을 사용하여 서명 확인 활성화로 설정해야 합니다.

프로세스

1. 명령줄에서 다음 명령을 실행하여 원하는 작업자에 대한 정보를 표시합니다.

```
$ oc describe machineconfigpool/worker
```

초기 작업자 모니터링의 출력 예

```
Name:         worker
Namespace:
Labels:       machineconfiguration.openshift.io/mco-built-in=
Annotations:  <none>
```

```
API Version:  machineconfiguration.openshift.io/v1
Kind:         MachineConfigPool
Metadata:
  Creation Timestamp:  2019-12-19T02:02:12Z
  Generation:          3
  Resource Version:    16229
  Self Link:           /apis/machineconfiguration.openshift.io/v1/machineconfigpools/worker
  UID:                 92697796-2203-11ea-b48c-fa163e3940e5
Spec:
  Configuration:
    Name:  rendered-worker-f6819366eb455a401c42f8d96ab25c02
    Source:
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         00-worker
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         01-worker-container-runtime
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         01-worker-kubelet
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         51-worker-rh-registry-trust
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         99-worker-ssh
  Machine Config Selector:
    Match Labels:
      machineconfiguration.openshift.io/role:  worker
  Node Selector:
    Match Labels:
      node-role.kubernetes.io/worker:
  Paused:                          false
Status:
  Conditions:
    Last Transition Time:  2019-12-19T02:03:27Z
    Message:
    Reason:
    Status:                False
    Type:                  RenderDegraded
    Last Transition Time:  2019-12-19T02:03:43Z
    Message:
    Reason:
    Status:                False
    Type:                  NodeDegraded
    Last Transition Time:  2019-12-19T02:03:43Z
    Message:
    Reason:
    Status:                False
    Type:                  Degraded
    Last Transition Time:  2019-12-19T02:28:23Z
    Message:
```

```
    Reason:
    Status:              False
    Type:                Updated
    Last Transition Time:  2019-12-19T02:28:23Z
    Message:             All nodes are updating to rendered-worker-
f6819366eb455a401c42f8d96ab25c02
    Reason:
    Status:              True
    Type:                Updating
  Configuration:
    Name:  rendered-worker-d9b3f4ffcfd65c30dcf591a0e8cf9b2e
    Source:
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                   00-worker
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                   01-worker-container-runtime
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                   01-worker-kubelet
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                   99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                   99-worker-ssh
  Degraded Machine Count:     0
  Machine Count:              1
  Observed Generation:        3
  Ready Machine Count:        0
  Unavailable Machine Count:  1
  Updated Machine Count:      0
Events:                       <none>
```

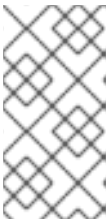
2. **oc describe** 명령을 다시 실행합니다.

```
$ oc describe machineconfigpool/worker
```

작업자가 업데이트된 후 출력 예

```
...
    Last Transition Time:  2019-12-19T04:53:09Z
    Message:             All nodes are updated with rendered-worker-
f6819366eb455a401c42f8d96ab25c02
    Reason:
    Status:              True
    Type:                Updated
    Last Transition Time:  2019-12-19T04:53:09Z
    Message:
    Reason:
    Status:              False
    Type:                Updating
  Configuration:
    Name:  rendered-worker-f6819366eb455a401c42f8d96ab25c02
```

```
    Source:
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 00-worker
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 01-worker-container-runtime
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 01-worker-kubelet
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 51-worker-rh-registry-trust
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 99-worker-ssh
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        4
  Ready Machine Count:        3
  Unavailable Machine Count:  0
  Updated Machine Count:      3
...
```



참고

Observed Generation 매개변수는 컨트롤러에서 생성된 구성의 생성에 따라 증가된 개수를 보여줍니다. 이 컨트롤러는 사양을 처리하고 수정본을 생성하지 못하더라도 이 값을 업데이트합니다. **Configuration Source** 값은 **51-worker-rh-registry-trust** 구성을 나타냅니다.

3. 다음 명령을 사용하여 **policy.json** 파일이 있는지 확인합니다.

```
$ oc debug node/<node> -- chroot /host cat /etc/containers/policy.json
```

출력 예

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
{
  "default": [
    {
      "type": "insecureAcceptAnything"
    }
  ],
  "transports": {
    "docker": {
      "registry.access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
```

```
            "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
          }
        ],
        "registry.redhat.io": [
          {
            "type": "signedBy",
            "keyType": "GPGKeys",
            "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
          }
        ]
      },
      "docker-daemon": {
        "": [
          {
            "type": "insecureAcceptAnything"
          }
        ]
      }
    }
}
```

4. 다음 명령을 사용하여 **registry.redhat.io.yaml** 파일이 있는지 확인합니다.

```
$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.redhat.io.yaml
```

출력 예

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
```

5. 다음 명령을 사용하여 **registry.access.redhat.yaml** 파일이 있는지 확인합니다.

```
$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.access.redhat.com.yaml
```

출력 예

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

2.4.3. 추가 리소스

- 머신 구성 개요

2.5. 컴플라이언스 이해

많은 OpenShift Container Platform 고객은 시스템을 프로덕션 환경에 도입하기 전에 일정 수준의 규제 준비 또는 컴플라이언스를 갖춰야 합니다. 이러한 규정 준비는 국가 표준, 산업 표준 또는 조직의 기업 거버넌스 프레임워크에 따라 규정될 수 있습니다.

2.5.1. 컴플라이언스 및 위험 관리 이해

FIPS 컴플라이언스는 보안 수준이 높은 환경에서 요구되는 가장 중요한 구성요소 중 하나로, 지원되는 암호화 기술만 노드에서 허용합니다.

중요

FIPS 검증 / 진행중인 모듈 암호화 라이브러리 사용은 **x86_64** 아키텍처의 OpenShift Container Platform 배포에서만 지원됩니다.

OpenShift Container Platform 컴플라이언스 프레임워크에 대한 Red Hat의 관점을 이해하려면 OpenShift 보안 가이드의 위험 관리 및 규제 준비 장을 참조하십시오.

추가 리소스

- FIPS 모드에서 클러스터 설치

2.6. 컨테이너 콘텐츠 보안

컨테이너 내부 콘텐츠의 보안을 유지하려면 Red Hat Universal Base Image와 같은 신뢰할 수 있는 기본 이미지로 시작하고 신뢰할 수 있는 소프트웨어를 추가해야 합니다. 컨테이너 이미지의 지속적인 보안을 확인하기 위해 이미지를 스캔하는 Red Hat 도구와 타사 도구가 있습니다.

2.6.1. 컨테이너 내부 보안

애플리케이션 및 인프라는 쉽게 사용할 수 있는 구성요소로 구성되며, 대부분은 Linux 운영 체제, JBoss Web Server, PostgreSQL 및 Node.js와 같은 오픈소스 패키지입니다.

이러한 패키지의 컨테이너화된 버전도 제공됩니다. 그러나 패키지의 원래 위치, 사용된 버전, 빌드한 사람 및 악성 코드가 있는지 여부를 알아야 합니다.

답변해야 할 몇 가지 질문은 다음과 같습니다.

- 컨테이너 내부의 구성요소 때문에 인프라가 손상됩니까?
- 애플리케이션 계층에 알려진 취약점이 있습니까?
- 런타임 및 운영 체제 계층이 최신입니까?

Red Hat UBI(Universal Base Images)의 컨테이너를 빌드하여 컨테이너 이미지의 기초가 Red Hat Enterprise Linux에 포함된 동일한 RPM 패키지 소프트웨어로 구성되게 합니다. UBI 이미지를 사용하거나 재배포하는 데 서브스크립션이 필요하지 않습니다.

컨테이너 자체의 지속적인 보안을 보장하기 위해 RHEL에서 직접 사용되거나 OpenShift Container Platform에 추가된 보안 스캔 기능을 통해 사용 중인 이미지에 취약점이 있을 때 경고할 수 있습니다. OpenSCAP 이미지 스캔은 RHEL에서 사용할 수 있으며 Red Hat Quay Container Security Operator 를 추가하여 OpenShift Container Platform에서 사용되는 컨테이너 이미지를 확인할 수 있습니다.

2.6.2. UBI를 사용하여 재배포 가능한 이미지 생성

컨테이너화된 애플리케이션을 생성하려면 일반적으로 운영 체제에서 제공하는 구성요소를 제공하는 신뢰할 수 있는 기본 이미지로 시작합니다. 여기에는 라이브러리, 유틸리티 및 운영 체제의 파일 시스템에서 애플리케이션에 표시될 것으로 예상되는 기타 기능이 포함됩니다.

Red Hat UBI(Universal Base Image)는 자체 컨테이너를 구축하는 모든 사용자가 Red Hat Enterprise Linux rpm 패키지 및 기타 콘텐츠를 사용하여 전적으로 구성된 컨테이너로 시작하도록 생성되었습니다. 이 UBI 이미지는 보안 패치로 최신 상태를 유지하고 고유 소프트웨어를 포함하도록 빌드된 컨테이너 이미지를 자유롭게 사용하고 재배포하도록 정기적으로 업데이트됩니다.

다른 UBI 이미지의 상태를 찾고 확인하려면 Red Hat Ecosystem Catalog 를 검색하십시오. 보안 컨테이너 이미지의 생성자는 다음과 같은 일반적인 두 가지 유형의 UBI 이미지에 관심이 있을 수 있습니다.

- **UBI**: RHEL 7 및 8용 표준 UBI 이미지(**ubi7/ubi 및 ubi8/ubi**) 외에도 해당 시스템(ubi**7/ubi-minimal 및 ubi8/ubi-mimimal**)을 기반으로 하는 최소 이미지도 있습니다. 이러한 이미지는 모두 표준 **yum** 및 **dnf** 명령을 사용하여 빌드한 컨테이너 이미지에 추가할 수 있는 RHEL 소프트웨어의 무료 리포지토리를 가리키도록 미리 구성되어 있습니다. Red Hat에서는 Fedora 및 Ubuntu와 같은 다른 배포판에서 이러한 이미지를 사용하도록 권장합니다.

- **Red Hat Software Collections**: Red Hat Ecosystem Catalog에서 **rhscl/** 을 검색하여 특정 유형의 애플리케이션에 대한 기본 이미지로 사용하기 위해 생성된 이미지를 찾습니다. 예를 들어, Apache httpd(**rhscl/httpd-***), Python(**rhscl/python-***), Ruby(**rhscl/ruby-***), Node.js(**rhscl/nodejs-***) 및 Perl(**rhscl/perl-***) rhscl 이미지가 있습니다.

UBI 이미지는 자유롭게 재배포할 수 있지만, 이러한 이미지에 관한 Red Hat 지원은 Red Hat 제품 서브스크립션을 통해서만 제공됩니다.

표준, 최소 및 초기화 UBI 이미지에서 사용하고 빌드하는 방법에 관한 정보는 Red Hat Enterprise Linux 설명서에서 Red Hat Universal Base 이미지 사용 을 참조하십시오.

2.6.3. RHEL의 보안 스캔

RHEL(Red Hat Enterprise Linux) 시스템의 경우 OpenSCAP 스캔은 **openscap-utils** 패키지에서 사용할 수 있습니다. RHEL에서 **openscap-podman** 명령을 사용하여 이미지의 취약점을 스캔할 수 있습니다. Red Hat Enterprise Linux 설명서에서 컨테이너 및 컨테이너 이미지에서 취약점 스캔 을 참조하십시오.

OpenShift Container Platform을 사용하면 CI/CD 프로세스에서 RHEL 스캐너를 활용할 수 있습니다. 예를 들어 소스 코드의 보안 결함을 테스트하는 정적 코드 분석 툴과 오픈소스 라이브러리에서 알려진 취약점과 같은 메타데이터를 제공하기 위해 해당 라이브러리르 식별하는 소프트웨어 구성 분석 툴을 통합할 수 있습니다.

2.6.3.1. OpenShift 이미지 스캔

OpenShift Container Platform에서 실행 중이고 Red Hat Quay 레지스트리에서 가져온 컨테이너 이미지의 경우 Operator를 사용하여 해당 이미지의 취약점을 나열할 수 있습니다. Red Hat Quay Container Security Operator 를 OpenShift Container Platform에 추가하여 선택한 네임스페이스에 추가된 이미지의 취약점 보고를 제공할 수 있습니다.

Red Hat Quay의 컨테이너 이미지 스캔은 Clair 보안 스캐너 를 사용하여 수행됩니다. Red Hat Quay에서 Clair는 RHEL, CentOS, Oracle, Alpine, Debian 및 Ubuntu 운영 체제 소프트웨어에서 빌드된 이미지의 취약점을 검색하고 보고할 수 있습니다.

2.6.4. 외부 스캔 통합

OpenShift Container Platform은 오브젝트 주석을 사용하여 기능을 확장합니다. 취약점 스캐너와 같은 외부 툴에서는 메타데이터로 이미지 오브젝트에 주석을 달아 결과를 요약하고 Pod 실행을 제어할 수 있습니다. 이 섹션에서는 이 주석의 인식된 형식을 설명하므로 유용한 데이터를 사용자에게 표시하기 위해 콘

솔에서 안정적으로 사용할 수 있습니다.

2.6.4.1. 이미지 메타데이터

패키지 취약점 및 오픈소스 소프트웨어(OSS) 라이센스 컴플라이언스를 포함하여 다양한 유형의 이미지 품질 데이터가 있습니다. 또한 이 메타데이터를 제공하는 공급자가 둘 이상일 수 있습니다. 이를 위해 다음 주석 형식이 예약되어 있습니다.

```
quality.images.openshift.io/<qualityType>.<providerId>: {}
```

표 2.1. 주석 키 형식

구성요소	설명	허용 가능한 값
qualityType	메타데이터 유형	**vulnerability** **license** **operations** **policy**
providerId	공급자 ID 문자열	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

2.6.4.1.1. 주석 키 예

```
quality.images.openshift.io/vulnerability.blackduck: {}
quality.images.openshift.io/vulnerability.jfrog: {}
quality.images.openshift.io/license.blackduck: {}
quality.images.openshift.io/vulnerability.openscap: {}
```

이미지 품질 주석의 값은 다음 형식을 준수해야 하는 구조화된 데이터입니다.

표 2.2. 주석 값 형식

필드	필수 여부	설명	유형
name	예	공급자 표시 이름	문자열
timestamp	예	스캔 타임스탬프	문자열
description	아니요	짧은 설명	문자열
reference	예	정보 소스 또는 자세한 내용의 URL. 사용자가 데이터를 검증하려면 필수	문자열
scannerVersion	아니요	스캐너 버전	문자열

필드	필수 여부	설명	유형
compliant	아니요	컴플라이언스 합격 또는 불합격	부울
summary	아니요	발견된 문제 요약	목록(아래 표 참조)

summary 필드는 다음 형식을 준수해야 합니다.

표 2.3. 요약 필드 값 형식

필드	설명	유형
label	구성요소의 표시 레이블(예: "심각", "중요", "중간", "낮음" 또는 "상태")	문자열
data	이 구성요소의 데이터(예: 발견된 취약점 수 또는 점수)	문자열
severityIndex	그래픽 표시의 순서를 지정하고 할당할 수 있는 구성요소 색인입니다. 값은 **0..3** 범위입니다. 여기서 **0** = 낮음입니다.	정수
reference	정보 소스 또는 자세한 내용의 URL. 선택 사항입니다.	문자열

2.6.4.1.2. 주석 값 예

이 예에서는 취약성 요약 데이터 및 컴플라이언스 부울이 있는 이미지의 OpenSCAP 주석을 표시합니다.

OpenSCAP 주석

```
{
  "name": "OpenSCAP",
  "description": "OpenSCAP vulnerability score",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://www.open-scap.org/930492",
  "compliant": true,
  "scannerVersion": "1.2",
  "summary": [
    { "label": "critical", "data": "4", "severityIndex": 3, "reference": null },
    { "label": "important", "data": "12", "severityIndex": 2, "reference": null },
    { "label": "moderate", "data": "8", "severityIndex": 1, "reference": null },
    { "label": "low", "data": "26", "severityIndex": 0, "reference": null }
  ]
}
```

이 예에서는 추가 세부 사항의 외부 URL이 있는 상태 인덱스 데이터가 있는 이미지의 Red Hat Ecosystem Catalog의 컨테이너 이미지 섹션 주석을 표시합니다.

Red Hat Ecosystem Catalog 주석

```
{
  "name": "Red Hat Ecosystem Catalog",
  "description": "Container health index",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://access.redhat.com/errata/RHBA-2016:1566",
  "compliant": null,
  "scannerVersion": "1.2",
  "summary": [
    { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null }
  ]
}
```

2.6.4.2. 이미지 오브젝트에 주석 달기

이미지 스트림 오브젝트는 OpenShift Container Platform의 최종 사용자가 작동하는 대상인 반면, 이미지 오브젝트는 보안 메타데이터로 주석을 답니다. 이미지 오브젝트는 클러스터 범위에 있으며, 여러 이미지 스트림 및 태그에서 참조할 수 있는 단일 이미지를 가리킵니다.

2.6.4.2.1. 주석 CLI 명령 예

<image>를 이미지 다이제스트로 교체합니다(예: **sha256:401e359e0f45bfdcf004e258b72e253fd07fba8cc5c6f2ed4f4608fb119ecc2**).

```
$ oc annotate image <image> \
    quality.images.openshift.io/vulnerability.redhatcatalog='{ \
    "name": "Red Hat Ecosystem Catalog", \
    "description": "Container health index", \
    "timestamp": "2020-06-01T05:04:46Z", \
    "compliant": null, \
    "scannerVersion": "1.2", \
    "reference": "https://access.redhat.com/errata/RHBA-2020:2347", \
    "summary": "[ \
      { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null } ]" }'
```

2.6.4.3. Pod 실행 제어

images.openshift.io/deny-execution 이미지 정책을 사용하여 이미지 실행 가능 여부를 프로그래밍 방식으로 제어합니다.

2.6.4.3.1. 주석 예

```
annotations:
  images.openshift.io/deny-execution: true
```

2.6.4.4. 통합 참조

대부분의 경우 취약성 스캐너와 같은 외부 툴은 이미지 업데이트를 감시하고 스캔을 수행하며 결과를 사용하여 관련 이미지 오브젝트에 주석을 추가하는 스크립트 또는 플러그인을 개발합니다. 일반적으로 이

자동화에서는 OpenShift Container Platform 4.6 REST API를 호출하여 주석을 작성합니다. REST API에 관한 일반 정보는 OpenShift Container Platform REST API를 참조하십시오.

2.6.4.4.1. REST API 호출 예

curl을 사용하는 다음 예제 호출은 주석의 값을 덮어씁니다. **<token>**, **<openshift_server>**, **<image_id>** 및 **<image_annotation>**의 값을 교체하십시오.

API 호출 패치

```
$ curl -X PATCH \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/merge-patch+json" \
  https://<openshift_server>:6443/apis/image.openshift.io/v1/images/<image_id> \
  --data '{ <image_annotation> }'
```

다음은 **PATCH** 페이로드 데이터의 예입니다.

호출 데이터 패치

```
{
"metadata": {
  "annotations": {
    "quality.images.openshift.io/vulnerability.redhatcatalog":
      "{ 'name': 'Red Hat Ecosystem Catalog', 'description': 'Container health index', 'timestamp': '2020-
06-01T05:04:46Z', 'compliant': null, 'reference': 'https://access.redhat.com/errata/RHBA-2020:2347',
'summary': [{'label': 'Health index', 'data': '4', 'severityIndex': 1, 'reference': null}] }"
    }
  }
}
```

추가 리소스

- 이미지 스트림 오브젝트

2.7. 안전하게 컨테이너 레지스트리 사용

컨테이너 레지스트리는 컨테이너 이미지를 다음에 저장합니다.

- 다른 사용자가 이미지에 액세스하도록 허용
- 이미지를 여러 버전의 이미지를 포함할 수 있는 리포지토리로 구성
- 선택적으로 다른 인증 방법을 기반으로 이미지에 대한 액세스를 제한하거나 공개적으로 사용 가능하게 합니다.

많은 사람과 조직이 이미지를 공유하는 Quay.io 및 Docker Hub와 같은 공용 컨테이너 레지스트리가 있습니다. Red Hat Registry에서는 지원되는 Red Hat 및 파트너 이미지를 제공하는 반면, Red Hat Ecosystem Catalog에서는 해당 이미지에 관한 자세한 설명 및 상태 점검을 제공합니다. 고유 레지스트리를 관리하기 위해 Red Hat Quay와 같은 컨테이너 레지스트리를 구매할 수 있습니다.

보안 관점에서 일부 레지스트리는 컨테이너의 상태를 확인하고 향상시키는 특수 기능을 제공합니다. 예를 들어, Red Hat Quay에서는 Clair 보안 스캐너를 사용한 컨테이너 취약성 스캔, GitHub 및 기타 위치에서 소스 코드가 변경될 때 이미지를 자동으로 재구성하는 빌드 트리거 및 역할 기반 액세스 제어(RBAC)를

사용하여 이미지에 대한 액세스를 보호하는 기능을 제공합니다.

2.7.1. 컨테이너의 출처를 알고 있습니까?

다운로드 및 배포된 컨테이너 이미지의 콘텐츠를 스캔하고 추적하는 데 사용할 수 있는 툴이 있습니다. 그러나 컨테이너 이미지의 공용 소스가 많이 있습니다. 공용 컨테이너 레지스트리를 사용하는 경우 신뢰할 수 있는 소스를 사용하여 보호 계층을 추가할 수 있습니다.

2.7.2. 불변의 인증된 컨테이너

*불변 컨테이너*를 관리할 때는 보안 업데이트를 사용하는 것이 특히 중요합니다. 불변 컨테이너는 실행 중에 변경되지 않는 컨테이너입니다. 불변 컨테이너를 배포할 때 하나 이상의 바이너리를 대체하기 위해 실행 중인 컨테이너로 들어가지 않습니다. 운영 관점에서 컨테이너를 변경하는 대신 업데이트된 컨테이너 이미지를 재빌드하고 재배포하여 컨테이너를 교체합니다.

Red Hat 인증 이미지는 다음과 같습니다.

- 플랫폼 구성 요소 또는 계층에 알려진 취약점이 없음
- 베어 메탈에서 클라우드까지 전체 RHEL 플랫폼에서 호환 가능
- Red Hat에서 지원

알려진 취약점 목록은 지속적으로 늘어나므로 시간 경과에 따라 배포된 컨테이너 이미지의 콘텐츠와 새로 다운로드한 이미지의 콘텐츠를 추적해야 합니다. RHSA(Red Hat Security Advisories)를 사용하여 Red Hat 인증 컨테이너 이미지에서 새로 발견된 문제를 경고하고 업데이트된 이미지로 안내할 수 있습니다. 또는 Red Hat Ecosystem Catalog로 이동하여 각 Red Hat 이미지에 관한 기타 보안 관련 문제를 조회할 수 있습니다.

2.7.3. Red Hat Registry 및 Ecosystem Catalog에서 컨테이너 가져오기

Red Hat에서는 Red Hat Ecosystem Catalog의 컨테이너 이미지 섹션에서 Red Hat 제품 및 파트너 제품의 인증된 컨테이너 이미지를 나열합니다. 해당 카탈로그에서 CVE, 소프트웨어 패키지 목록 및 상태 점수를 포함하여 각 이미지의 세부 정보를 볼 수 있습니다.

Red Hat 이미지는 실제로 공개 컨테이너 레지스트리(**registry.access.redhat.com**) 및 인증된 레지스트리(**registry.redhat.io**)로 표시되는 *Red Hat Registry*에 저장됩니다. 둘 다 기본적으로 동일한 컨테이너 이미지 세트를 포함하며 **registry.redhat.io**에는 Red Hat 서브스크립션 인증서로 인증해야 하는 추가 이미지가 포함됩니다.

Red Hat에서 컨테이너 콘텐츠에 취약점이 있는지 모니터링하고 정기적으로 업데이트합니다. Red Hat에서 *glibc*, DROWN 또는 Dirty Cow에 대한 수정 사항과 같은 보안 업데이트를 릴리스하면 영향을 받는 컨테이너 이미지도 다시 빌드되어 Red Hat Registry에 푸시됩니다.

Red Hat에서는 **상태 색인**을 사용하여 Red Hat Ecosystem Catalog를 통해 제공되는 각 컨테이너의 보안 위험을 반영합니다. 컨테이너에서는 Red Hat과 에라타 프로세스에서 제공하는 소프트웨어를 사용하므로 오래되어 해지된 컨테이너는 안전하지 않은 반면 새로운 컨테이너는 더 안전합니다.

컨테이너의 수명을 설명하기 위해 Red Hat Ecosystem Catalog에서는 평가 시스템을 사용합니다. 최신 등급은 이미지에 사용 가능한 가장 오래되고 가장 심각한 보안 정오표 수치입니다. "A"는 "F"보다 최신입니다. 이 평가 시스템에 관한 자세한 내용은 Red Hat Ecosystem Catalog 내부에서 사용되는 컨테이너 상태 색인 등급을 참조하십시오.

Red Hat 소프트웨어와 관련된 보안 업데이트 및 취약점에 관한 자세한 내용은 Red Hat 제품 보안 센터를 참조하십시오. Red Hat Security Advisories를 확인하여 특정 권고와 CVE를 검색하십시오.

2.7.4. OpenShift Container Registry

OpenShift Container Platform에는 컨테이너 이미지를 관리하는 데 사용할 수 있는 플랫폼의 통합 구성요소로 실행되는 프라이빗 레지스트리인 *OpenShift Container Registry*가 포함되어 있습니다. OpenShift Container Registry에서는 누가 어떤 컨테이너 이미지를 가져오고 푸시하는지 관리할 수 있는 역할 기반 액세스 제어를 제공합니다.

OpenShift Container Platform에서는 Red Hat Quay와 같이 이미 사용 중인 다른 프라이빗 레지스트리와의 통합도 지원합니다.

추가 리소스

- 통합된 OpenShift Container Platform 레지스트리

2.7.5. Red Hat Quay를 사용하여 컨테이너 저장

Red Hat Quay는 Red Hat에서 제공하는 엔터프라이즈급 컨테이너 레지스트리 제품입니다. Red Hat Quay의 개발은 업스트림 프로젝트 Quay를 통해 수행됩니다. Red Hat Quay는 온프레미스 또는 Quay.io에서 호스팅된 Red Hat Quay 버전을 통해 배포할 수 있습니다.

Red Hat Quay의 보안 관련 기능은 다음과 같습니다.

- **타임 머신**: 오래된 태그가 있는 이미지가 설정된 기간이 지난 후 만료되거나 사용자가 선택한 만료 시간을 기준으로 만료될 수 있습니다.
- **저장소 미러링**: 회사 방화벽 뒤에서 Red Hat Quay에서 공용 리포지토리를 호스팅하거나 성능상의 이유로 레지스트리가 사용되는 위치와 더 가까운 레지스트리를 호스팅하는 등 보안상의 이유로 다른 레지스트리를 미러링할 수 있습니다.
- **작업 로그 스토리지**: 나중에 검색 및 분석할 수 있도록 Red Hat Quay 로깅 출력을 Elasticsearch 스토리지에 저장합니다.
- **Clair 보안 스캔**: 각 컨테이너 이미지의 출처에 따라 다양한 Linux 취약점 데이터베이스와 비교하여 이미지를 스캔합니다.
- **내부 인증**: 기본 로컬 데이터베이스를 사용하여 Red Hat Quay에 대한 RBAC 인증을 처리하거나 LDAP, Keystone(OpenStack), JWT 사용자 정의 인증 또는 외부 애플리케이션 토큰 인증 중에서 선택합니다.
- **외부 인증 (OAuth)**: GitHub, GitHub Enterprise 또는 Google Authentication에서 Red Hat Quay에 대한 권한 부여 허용.
- **액세스 설정**: docker, rkt, 익명 액세스, 사용자 생성 계정, 암호화된 클라이언트 암호 또는 접두사 사용자 이름 자동 완성에서 Red Hat Quay에 액세스할 수 있도록 토큰을 생성합니다.

OpenShift Container Platform과 Red Hat Quay는 지속적으로 통합되며, 특히 흥미로운 OpenShift Container Platform Operators가 여러 개 있습니다. Quay Bridge Operator를 사용하면 내부 OpenShift Container Platform 레지스트리를 Red Hat Quay로 교체할 수 있습니다. Quay Red Hat Quay Container Security Operator 를 사용하면 Red Hat Quay 레지스트리에서 가져온 OpenShift Container Platform에서 실행중인 이미지의 취약점을 확인할 수 있습니다.

2.8. 빌드 프로세스 보안

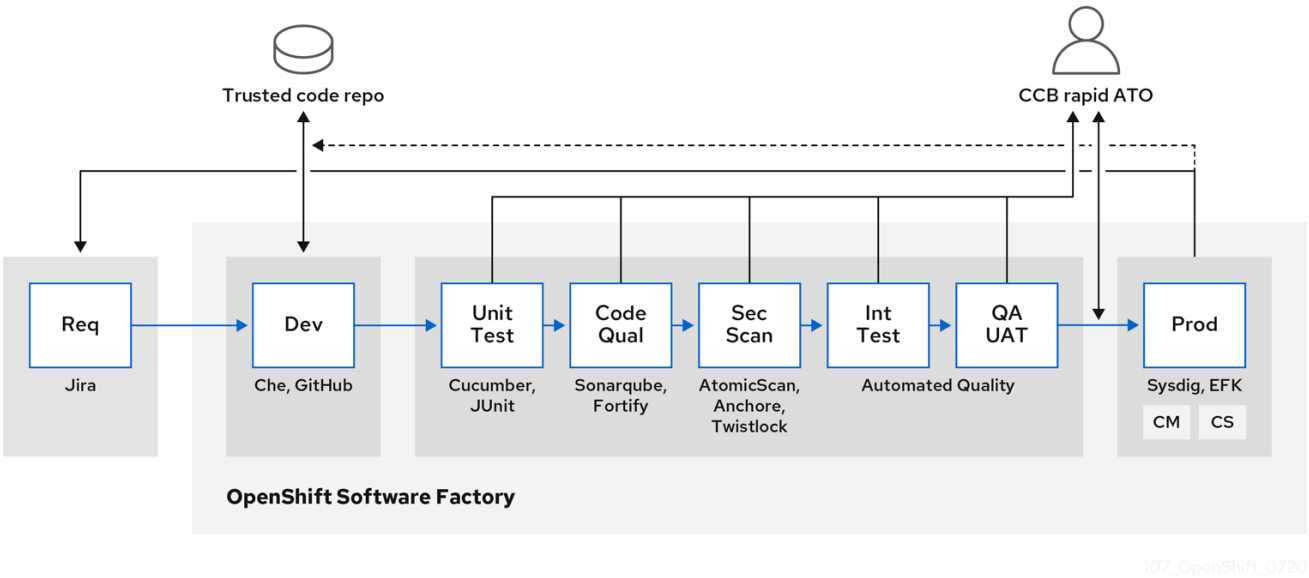
컨테이너 환경에서 소프트웨어 빌드 프로세스는 애플리케이션 코드가 필수 런타임 라이브러리와 통합되는 라이프사이클의 단계입니다. 이 빌드 프로세스 관리는 소프트웨어 스택을 보호하는 데 핵심입니다.

2.8.1. 한 번 빌드하여 어디에나 배포

컨테이너 빌드를 위한 표준 플랫폼으로 OpenShift Container Platform을 사용하면 빌드 환경의 보안을 보장할 수 있습니다. "한 번만 빌드하여 어디에나 배포" 철학을 준수하면 빌드 프로세스의 제품을 프로덕션에 정확히 배포할 수 있습니다.

컨테이너의 불변성을 유지 관리하는 것도 중요합니다. 실행 중인 컨테이너에 패치를 적용하지 말고 다시 빌드하여 재배치해야 합니다.

소프트웨어가 빌드, 테스트 및 프로덕션 단계를 진행해 갈 때 소프트웨어 공급망을 구성하는 툴을 신뢰하는 것이 중요합니다. 다음 그림에서는 컨테이너화된 소프트웨어를 위한 신뢰할 수 있는 소프트웨어 공급망에 통합될 수 있는 프로세스와 툴을 보여줍니다.



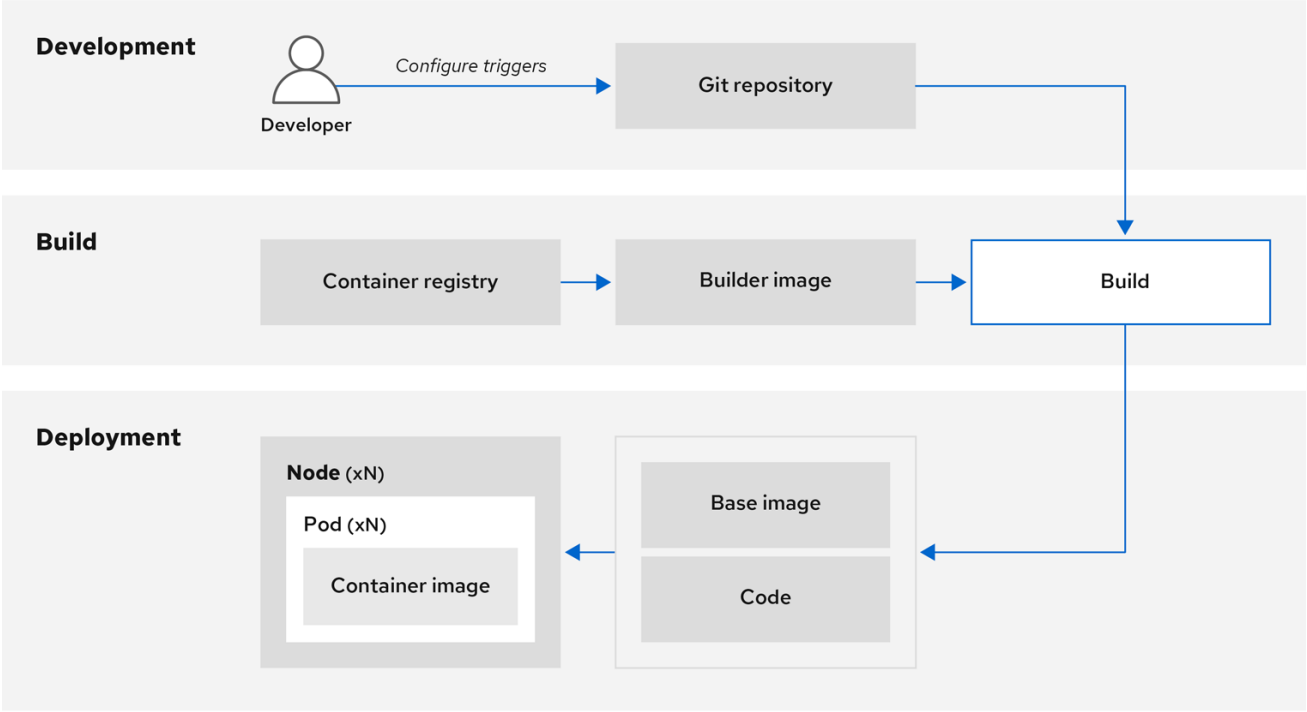
OpenShift Container Platform은 신뢰할 수 있는 코드 리포지토리(예: GitHub) 및 개발 플랫폼(예: Che)과 통합되어 보안 코드를 생성하고 관리할 수 있습니다. 단위 테스트에서는 Cucumber 및 JUnit을 사용합니다. 컨테이너에서 Anchore 또는 Twistlock의 취약점 및 컴플라이언스 문제가 있는지 검사하고 AtomicScan 또는 Clair와 같은 이미지 검색 툴을 사용할 수 있습니다. Sysdig와 같은 툴을 사용하면 컨테이너화된 애플리케이션을 지속적으로 모니터링할 수 있습니다.

2.8.2. 빌드 관리

S2I(Source-to-Image)를 사용하여 소스 코드와 기본 이미지를 결합할 수 있습니다. *빌더 이미지*는 S2I를 사용하므로 개발 및 운영 팀이 재현 가능한 빌드 환경에서 협업할 수 있습니다. UBI(Universal Base Image) 이미지로 사용 가능한 Red Hat S2I 이미지를 사용하면 실제 RHEL RPM 패키지에서 빌드된 기본 이미지로 소프트웨어를 자유롭게 재배포할 수 있습니다. Red Hat에서는 이를 허용하기 위해 서브스크립션 제한 사항을 제거했습니다.

개발자가 빌드 이미지를 사용하여 애플리케이션용 Git로 코드를 커밋하면 OpenShift Container Platform에서는 다음 기능을 수행할 수 있습니다.

- 사용 가능한 아티팩트, S2I 빌더 이미지 및 새로 커밋된 코드에서 자동으로 새 이미지를 어셈블링하기 위해 코드 리포지토리의 웹 후크를 사용하거나 기타 자동 연속 통합(CI) 프로세스를 사용하여 트리거합니다.
- 테스트를 위해 새로 빌드된 이미지를 자동으로 배포합니다.
- 테스트된 이미지를 CI 프로세스를 사용하여 자동으로 배포할 수 있는 프로덕션으로 승격합니다.



통합 OpenShift Container Registry를 사용하여 최종 이미지에 대한 액세스를 관리할 수 있습니다. S2I 및 기본 빌드 이미지가 자동으로 OpenShift Container Registry로 푸시됩니다.

포함된 Jenkins for CI 외에도 RESTful API를 사용하여 자체 빌드 및 CI 환경을 OpenShift Container Platform과 통합하고 API 호환 이미지 레지스트리를 사용할 수 있습니다.

2.8.3. 빌드 중 입력 보안

일부 시나리오에서는 빌드 작업을 할 때 종속 리소스에 액세스하기 위한 인증서가 필요하지만 빌드에서 생성한 최종 애플리케이션 이미지에서 해당 인증서가 사용 가능한 것은 바람직하지 않습니다. 이 목적을 위해 입력 보안을 정의할 수 있습니다.

예를 들어 Node.js 애플리케이션을 빌드할 때 Node.js 모듈에 맞게 개인용 미러를 설정할 수 있습니다. 이 개인용 미러에서 모듈을 다운로드하려면 URL, 사용자 이름 및 암호가 포함된 빌드에 사용할 사용자 정의 **.npmrc** 파일을 제공해야 합니다. 보안상의 이유로 애플리케이션 이미지에 인증서를 노출하지 않아야 합니다.

이 예제 시나리오를 사용하여 새 **BuildConfig** 오브젝트에 입력 보안을 추가할 수 있습니다.

1. 보안이 없으면 다음과 같이 생성합니다.

```
$ oc create secret generic secret-npmrc --from-file=.npmrc=~/.npmrc
```

그러면 **~/.npmrc** 파일의 base64 인코딩 콘텐츠를 포함하는 **secret-npmrc**라는 새 보안이 생성됩니다.

2. 다음과 같이 기존 **BuildConfig** 오브젝트의 **source** 섹션에 보안을 추가합니다.

```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
```

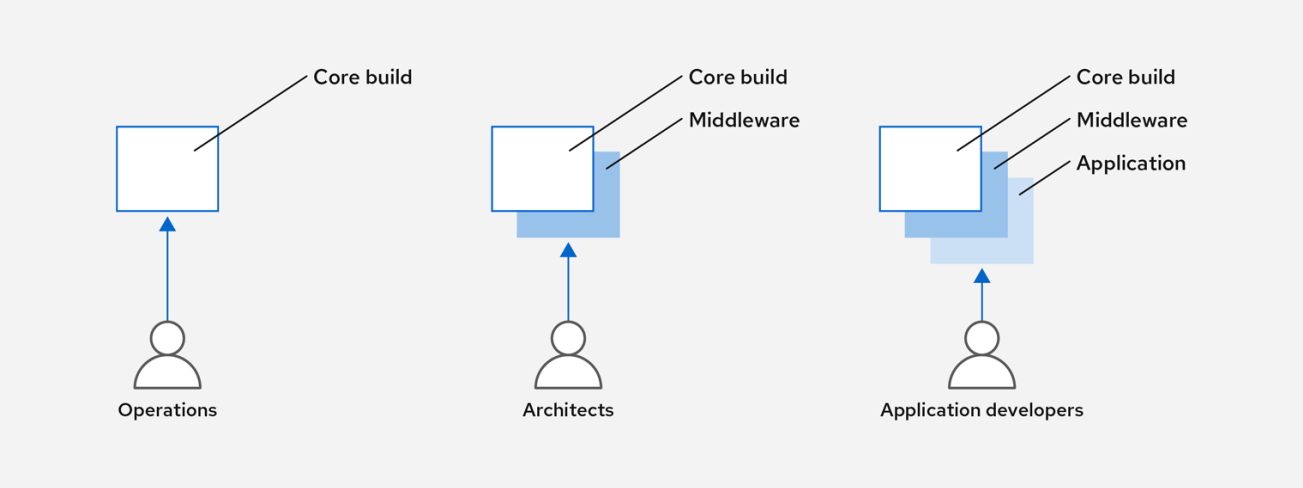
```
- destinationDir: .
  secret:
    name: secret-npmrc
```

3. 새 **BuildConfig** 오브젝트에 보안을 포함하려면 다음 명령을 실행합니다.

```
$ oc new-build \
    openshift/nodejs-010-centos7~https://github.com/sclorg/nodejs-ex.git \
    --build-secret secret-npmrc
```

2.8.4. 빌드 프로세스 설계

개별적으로 제어할 수 있도록 컨테이너 이미지 관리를 설계하고 컨테이너 계층을 사용하도록 프로세스를 구축할 수 있습니다.



예를 들어, 운영 팀은 기본 이미지를 관리하는 반면 아키텍트는 미들웨어, 런타임, 데이터베이스 및 기타 솔루션을 관리합니다. 그런 다음 개발자는 애플리케이션 계층과 코드 작성에 중점을 둘 수 있습니다.

매일 새로운 취약점이 식별되므로 시간 경과에 따라 컨테이너 콘텐츠를 사전 대응식으로 확인해야 합니다. 이를 위해서는 자동화된 보안 테스트를 빌드 또는 CI 프로세스에 통합해야 합니다. 예를 들면 다음과 같습니다.

- SAST/DAST – 정적 및 동적 보안 테스트 도구.
- 알려진 취약점과 비교하여 실시간 검사를 위한 스캐너. 이러한 툴을 통해서는 컨테이너의 오픈소스 패키지를 카탈로그화하고 알려진 취약점을 사용자에게 알리며 이전에 스캔한 패키지에서 새로운 취약점이 발견되면 사용자에게 이 내용을 업데이트합니다.

CI 프로세스에는 보안 스캔에서 발견된 문제로 빌드에 플래그를 표시하는 정책이 포함되어 있으므로, 팀이 해당 문제를 해결하기 위해 적절한 조치를 취할 수 있습니다. 빌드와 배포 사이에 아무것도 변경되지 않도록 사용자 정의 빌드 컨테이너에 서명해야 합니다.

GitOps 방법론을 사용하면 동일한 CI/CD 메커니즘을 사용하여 애플리케이션 구성뿐만 아니라 OpenShift Container Platform 인프라를 관리할 수 있습니다.

2.8.5. Knative 서버리스 애플리케이션 빌드

Kubernetes 및 Kourier를 사용하면 OpenShift Container Platform에서 OpenShift Serverless를 사용하여 서버리스 애플리케이션을 구축, 배포 및 관리할 수 있습니다.

다른 빌드와 마찬가지로 S2I 이미지를 사용하여 컨테이너를 빌드한 다음 Knative 서비스를 사용하여 컨테이너를 제공할 수 있습니다. OpenShift Container Platform 웹 콘솔의 **토폴로지** 보기를 통해 Knative 애플리케이션 빌드를 봅니다.

2.8.6. 추가 리소스

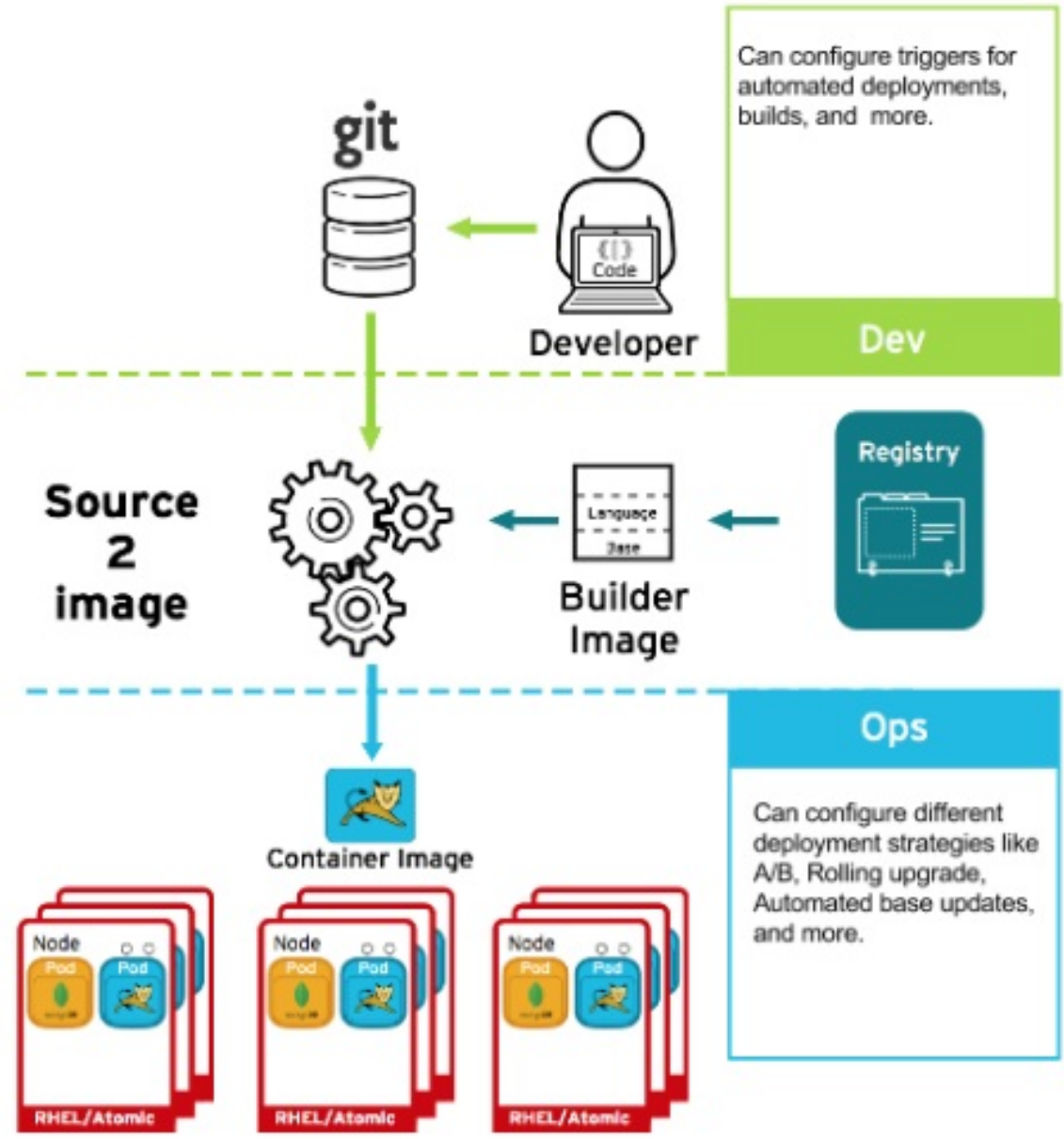
- 이미지 빌드 이해
- 빌드 트리거 및 수정
- 빌드 입력 생성
- 입력 보안 및 구성 맵
- CI/CD 방법론 및 관행
- About OpenShift Serverless
- 토폴로지 보기를 사용하여 애플리케이션 구성 보기

2.9. 컨테이너 배포

사용자가 배포한 컨테이너가 최신 프로덕션 품질의 콘텐츠를 보유하고 있으며 변경되지 않았는지 다양한 기술을 사용하여 확인할 수 있습니다. 이러한 기술에는 최신 코드를 통합하도록 빌드 트리거를 설정하는 것과 서명을 사용하여 컨테이너가 신뢰할 수 있는 소스에서 제공되었으며 수정되지 않았는지 확인하는 것이 포함됩니다.

2.9.1. 트리거를 사용하여 컨테이너 배포 제어

빌드 프로세스 중에 문제가 발생하거나 이미지가 배포된 후 취약점이 발견되면 자동화된 정책 기반 배포 도구를 사용하여 문제를 해결할 수 있습니다. 실행 중인 컨테이너에 패치를 적용하는 것이 아니라, 트리거를 사용하여 이미지를 다시 빌드하고 교체함으로써 컨테이너 프로세스가 변경되지 않게 할 수 있습니다.



예를 들어 코어, 미들웨어 및 애플리케이션의 세 가지 컨테이너 이미지 계층을 사용하여 애플리케이션을 빌드합니다. 핵심 이미지에서 문제가 발견되고 해당 이미지가 다시 빌드됩니다. 빌드가 완료되면 이미지가 OpenShift Container Registry로 푸시됩니다. OpenShift Container Platform에서 이미지가 변경되었음을 감지하고 정의된 트리거를 기반으로 애플리케이션 이미지를 자동으로 재빌드하고 배포합니다. 이 변경은 고정 라이브러리를 통합하고 프로덕션 코드가 최신 이미지와 동일하게 합니다.

oc set triggers 명령을 사용하여 배포 트리거를 설정할 수 있습니다. 예를 들어 deployment-example이라는 배포용 트리거를 설정하려면 다음을 수행합니다.

```
$ oc set triggers deploy/deployment-example \
    --from-image=example:latest \
    --containers=web
```

2.9.2. 배포할 수 있는 이미지 소스 제어

원하는 이미지가 실제로 배포되고 포함된 콘텐츠를 포함하는 이미지가 신뢰할 수 있는 소스의 이미지이며 변경되지 않은 것이 중요합니다. 암호화 서명을 사용하면 이를 보장할 수 있습니다. OpenShift Container Platform을 사용하면 클러스터 관리자는 배포 환경 및 보안 요구 사항을 반영하여 광범위하거나 한정된 보안 정책을 적용할 수 있습니다. 이 정책을 정의하는 매개변수는 다음 두 가지입니다.

- 선택적 프로젝트 네임스페이스가 있는 레지스트리 하나 이상
- 공개 키 수락, 거부 또는 필수와 같은 신뢰 유형

이러한 정책 매개변수를 사용하여 전체 레지스트리, 레지스트리 일부 또는 개별 이미지의 신뢰 관계를 허용, 거부 또는 필수로 지정할 수 있습니다. 신뢰할 수 있는 공개 키를 사용하면 암호화 방식으로 소스를 확인할 수 있습니다. 정책 규칙은 노드에 적용됩니다. 정책은 모든 노드에 균일하게 적용되거나 다른 노드 워크로드(예 : 빌드, 영역 또는 환경)를 대상으로 할 수 있습니다.

이미지 서명 정책 파일 예

```
{
    "default": [{"type": "reject"}],
    "transports": {
        "docker": {
            "access.redhat.com": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ]
        },
        "atomic": {
            "172.30.1.1:5000/openshift": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ],
            "172.30.1.1:5000/production": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/example.com/pubkey"
                }
            ],
            "172.30.1.1:5000": [{"type": "reject"}]
        }
    }
}
```

정책은 **/etc/containers/policy.json**으로 노드에 저장될 수 있습니다. 이 파일을 노드에 저장할 때는 새로운 **MachineConfig** 오브젝트를 사용하는 것이 가장 좋습니다. 이 예에서는 다음 규칙을 적용합니다.

- Red Hat 공개 키로 Red Hat Registry(**registry.access.redhat.com**)의 이미지에 서명해야 합니다.
- **openshift** 네임스페이스에 있는 OpenShift Container Registry의 이미지에 Red Hat 공개 키로 서명해야 합니다.
- **production** 네임스페이스에 있는 OpenShift Container Registry의 이미지에 **example.com**의 공개 키로 서명해야 합니다.
- 글로벌 **default** 정의로 지정되지 않은 다른 모든 레지스트리는 거부됩니다.

2.9.3. 서명 전송 사용

서명 전송은 바이너리 서명 Blob을 저장하고 검색하는 방법입니다. 서명 전송의 유형은 다음 두 가지입니다.

- **atomic**: OpenShift Container Platform API에서 관리합니다.
- **docker**: 로컬 파일 또는 웹 서버에서 제공.

OpenShift Container Platform API는 **atomic** 전송 유형을 사용하는 서명을 관리합니다. 이 서명 유형을 사용하는 이미지를 OpenShift 컨테이너 레지스트리에 저장해야 합니다. docker/distribution **extensions** API에서 이미지 서명 끝점을 자동 검색하므로 추가 구성이 필요하지 않습니다.

docker 전송 유형을 사용하는 서명은 로컬 파일 또는 웹 서버에서 제공합니다. 이 서명은 더 유연합니다. 컨테이너 이미지 레지스트리에서 이미지를 제공하고 독립 서버를 사용하여 바이너리 서명을 제공할 수 있습니다.

그러나 **docker** 전송 유형에는 추가 구성이 필요합니다. 임의로 이름이 지정된 YAML 파일을 기본적으로 호스트 시스템의 디렉터리인 **/etc/containers/registries.d**에 배치하여 서명 서버의 URI로 노드를 구성해야 합니다. YAML 구성 파일에는 레지스트리 URI 및 서명 서버 URI 또는 *sigstore*가 포함되어 있습니다.

registries.d 파일 예

```
docker:
    access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

이 예에서 Red Hat Registry, **access.redhat.com**은 **docker** 전송 유형의 서명을 제공하는 서명 서버입니다. 해당 URI는 **sigstore** 매개변수에 정의되어 있습니다. 이 파일의 이름을 **/etc/containers/registries.d/redhat.com.yam**로 지정하고 Machine Config Operator를 사용하여 파일을 클러스터의 각 노드에 자동으로 배치합니다. 정책 및 **registries.d** 파일은 컨테이너 런타임을 통해 동적으로 로드되므로 서비스를 다시 시작할 필요가 없습니다.

2.9.4. 보안 및 구성 맵 생성

Secret 오브젝트 유형에서는 암호, OpenShift Container Platform 클라이언트 구성 파일, **dockercfg** 파일, 개인 소스 리포지토리 인증서와 같은 중요한 정보를 보유하는 메커니즘을 제공합니다. 보안은 Pod에서 민감한 콘텐츠를 분리합니다. 볼륨 플러그인을 사용하여 컨테이너에 보안을 마운트하거나 시스템에서 보안을 사용하여 Pod 대신 조치를 수행할 수 있습니다.

예를 들어 개인 이미지 리포지토리에 액세스할 수 있도록 배치 구성에 보안을 추가하려면 다음을 수행하십시오.

프로세스

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. 새 프로젝트를 생성합니다.
3. **리소스 → 보안**으로 이동하여 새 보안을 생성합니다. **보안 유형**을 **이미지 보안**으로 설정하고 **인증 유형**을 **이미지 레지스트리 인증서**로 설정하여 개인 이미지 리포지토리에 액세스하는 데 사용할 인증서를 입력합니다.
4. 배포 구성을 생성할 때(예: **프로젝트에 추가 → 이미지 배포** 페이지) **가져오기 보안**을 새 보안으로 설정합니다.

구성 맵은 보안과 유사하지만 민감한 정보가 포함되지 않은 문자열 작업을 지원하도록 설계되었습니다. **ConfigMap** 오브젝트에는 Pod에서 사용하거나 컨트롤러와 같은 시스템 구성 요소의 구성 데이터를 저장하는 데 사용할 수 있는 구성 데이터의 키-값 쌍이 있습니다.

2.9.5. 지속적인 배포 자동화

OpenShift Container Platform과 연속 배포(CD) 툴링을 통합할 수 있습니다.

CI/CD 및 OpenShift Container Platform을 활용하면 최신 수정 사항을 통합하기 위해 애플리케이션을 재빌드하고 테스트한 다음 환경 내 모든 위치에 배포되었는지 확인하는 프로세스를 자동화할 수 있습니다.

추가 리소스

- 입력 보안 및 구성 맵

2.10. 컨테이너 플랫폼 보안

OpenShift Container Platform 및 쿠버네티스 API는 대규모 컨테이너 관리 자동화의 핵심입니다. API는 다음을 수행하는 데 사용됩니다.

- Pod, 서비스 및 복제 컨트롤러의 데이터를 검증하고 구성합니다.
- 수신 요청에서 프로젝트의 유효성을 확인하고 다른 주요 시스템 구성요소에서 트리거를 호출합니다.

쿠버네티스를 기반으로 하는 OpenShift Container Platform의 보안 관련 기능은 다음과 같습니다.

- 역할 기반 액세스 제어 및 네트워크 정책을 결합하여 컨테이너를 여러 수준으로 격리하는 멀티 테넌시.
- API와 API에 요청하는 주체 사이의 경계를 형성하는 승인 플러그인.

OpenShift Container Platform에서는 Operator를 사용하여 쿠버네티스 수준 보안 기능 관리를 자동화하고 단순화합니다.

2.10.1. 멀티 테넌시로 컨테이너 격리

다중 테넌트를 사용하면 여러 사용자가 소유하고 여러 호스트와 네임스페이스에서 실행되는 OpenShift Container Platform 클러스터의 애플리케이션을 서로 분리하고 외부 공격으로부터 격리된 상태로 유지할 수 있습니다. 쿠버네티스 네임스페이스에 역할 기반 액세스 제어(RBAC)를 적용하여 멀티 테넌시를 확보합니다.

쿠버네티스에서 *네임스페이스*는 다른 애플리케이션과 분리된 방식으로 애플리케이션을 실행할 수 있는 영역입니다. OpenShift Container Platform에서는 SELinux에서 MCS 레이블링을 포함하여 주석을 추가하고 이러한 확장 네임스페이스를 *프로젝트*로 식별하여 네임스페이스를 사용하고 확장합니다. 프로젝트 범위 내에서 서비스 계정, 정책, 제약 조건 및 기타 다양한 오브젝트를 포함하여 자체 클러스터 리소스를 유지 관리할 수 있습니다.

선택된 사용자가 프로젝트에 액세스할 수 있도록 RBAC 오브젝트가 해당 프로젝트에 할당됩니다. 이 인증에서는 규칙, 역할 및 바인딩 양식을 사용합니다.

- 규칙을 통해서는 사용자가 프로젝트에서 생성하거나 액세스할 수 있는 대상을 정의합니다.
- 역할은 선택한 사용자 또는 그룹에 바인딩할 수 있는 규칙 컬렉션입니다.

- 바인딩을 통해서는 사용자 또는 그룹과 역할 간의 연결을 정의합니다.

로컬 RBAC 역할 및 바인딩은 사용자 또는 그룹을 특정 프로젝트에 연결합니다. 클러스터 RBAC는 클러스터 전체 역할 및 바인딩을 클러스터의 모든 프로젝트에 연결할 수 있습니다. **admin**, **basic-user**, **cluster-admin** 및 **cluster-status** 액세스를 제공하기 위해 지정할 수 있는 기본 클러스터 역할이 있습니다.

2.10.2. 승인 플러그인으로 컨트롤 플레인 보호

RBAC는 사용자와 그룹 및 사용 가능한 프로젝트 간의 액세스 규칙을 제어하지만 *승인 플러그인*은 OpenShift Container Platform 마스터 API에 대한 액세스를 정의합니다. 승인 플러그인은 다음과 같은 일련의 규칙으로 구성됩니다.

- 기본 승인 플러그인: 이는 OpenShift Container Platform 컨트롤 플레인의 구성 요소에 적용되는 기본 정책 및 리소스 제한 세트를 구현합니다.
- 변경 승인 플러그인: 이러한 플러그인은 동적으로 승인 체인을 확장합니다. 이 플러그인은 웹 후크 서버를 호출하고 요청을 인증하며 선택된 리소스를 수정할 수 있습니다.
- 승인 플러그인 검증: 이러한 경우 선택한 리소스에 대한 요청의 유효성을 검사하고 요청의 유효성을 검사하고 리소스가 다시 변경되지 않는지 확인할 수 있습니다.

API 요청이 체인의 승인 플러그인을 진행하는 중에 실패가 발생하여 요청이 거부됩니다. 각 승인 플러그인은 특정 리소스와 연관되어 있으며 해당 리소스의 요청에만 응답합니다.

2.10.2.1. SCC(보안 컨텍스트 제약 조건)

SCC(*보안 컨텍스트 제약 조건*)를 사용하여 시스템에 적용하기 위해 Pod를 실행해야 하는 조건 집합을 정의할 수 있습니다.

SCC에서 관리할 수 있는 몇 가지 요소는 다음과 같습니다.

- 권한이 있는 컨테이너 실행
- 컨테이너가 추가하도록 요청할 수 있는 기능
- 호스트 디렉터리를 볼륨으로 사용
- 컨테이너의 SELinux 컨텍스트
- 컨테이너 사용자 ID

필수 권한이 있으면 필요한 경우 기본 SCC 정책의 허용 범위를 넓게 조정할 수 있습니다.

2.10.2.2. 서비스 계정에 역할 부여

사용자에게 역할 기반 액세스 권한이 할당된 방식과 동일하게 서비스 계정에 역할을 할당할 수 있습니다. 프로젝트마다 3개의 기본 서비스 계정이 생성됩니다. 서비스 계정:

- 특정 프로젝트로 범위가 제한됨
- 프로젝트에서 이름 파생
- OpenShift Container Registry에 액세스하기 위해 API 토큰 및 인증서가 자동으로 할당됨

플랫폼 구성요소와 관련된 서비스 계정의 키는 자동으로 순환됩니다.

2.10.3. 인증 및 권한 부여

2.10.3.1. OAuth를 사용하여 액세스 제어

컨테이너 플랫폼 보안을 위해 인증 및 권한 부여를 통해 API 액세스 제어를 사용할 수 있습니다. OpenShift Container Platform 마스터에는 내장 OAuth 서버가 포함되어 있습니다. 사용자가 API에 자신을 인증하기 위해 OAuth 액세스 토큰을 가져올 수 있습니다.

관리자는 LDAP, GitHub 또는 Google과 같은 *ID 공급자*를 사용하여 인증할 OAuth를 구성할 수 있습니다. ID 공급자는 기본적으로 새로운 OpenShift Container Platform 배포에 사용되지만 초기 설치 시 또는 설치 후 이 공급자를 구성할 수 있습니다.

2.10.3.2. API 액세스 제어 및 관리

애플리케이션에는 관리가 필요한 끝점이 서로 다른 여러 개의 독립적인 API 서비스가 있을 수 있습니다. OpenShift Container Platform에는 3scale API 게이트웨이의 컨테이너화된 버전이 포함되어 있으므로 API를 관리하고 액세스를 제어할 수 있습니다.

3scale에서는 API 인증 및 보안을 위한 다양한 표준 옵션을 제공합니다. 이 옵션은 표준 API 키, 애플리케이션 ID와 키 쌍 및 OAuth 2.0과 함께 인증서를 발행하고 액세스를 제어하기 위해 조합하여 사용하거나 단독으로 사용할 수 있습니다.

특정 끝점, 방법 및 서비스에 대한 액세스를 제한하고 사용자 그룹의 액세스 정책을 적용할 수 있습니다. 애플리케이션 계획을 통해 API 사용에 대한 속도 제한을 설정하고 개발자 그룹의 트래픽 흐름을 제어할 수 있습니다.

컨테이너화된 3scale API 게이트웨이인 APIcast v2를 사용하는 방법에 관한 자습서는 3scale 설명서의 Red Hat OpenShift에서 APIcast 실행 을 참조하십시오.

2.10.3.3. Red Hat Single Sign-On

Red Hat Single Sign-On 서버를 사용하면 SAML 2.0, OpenID Connect 및 OAuth 2.0을 포함한 표준을 기반으로 웹 싱글 사인온 기능을 제공하여 애플리케이션을 보호할 수 있습니다. 서버는 SAML 또는 OpenID Connect 기반 ID 공급자(IdP)의 역할을 하며 엔터프라이즈 사용자 디렉터리 또는 타사 ID 공급자와 함께 표준 기반 토큰을 사용하여 ID 정보 및 애플리케이션을 중재할 수 있습니다. Red Hat Single Sign-On을 Microsoft Active Directory 및 Red Hat Enterprise Linux Identity Management를 포함한 LDAP 기반 디렉터리 서비스와 통합할 수 있습니다.

2.10.3.4. 보안 셀프 서비스 웹 콘솔

OpenShift Container Platform에서는 팀이 권한없이 다른 환경에 액세스하지 못하도록 셀프 서비스 웹 콘솔을 제공합니다. OpenShift Container Platform에서는 다음을 제공하여 보안 멀티 테넌트 마스터를 보장합니다.

- 마스터에 액세스하는 데 TLS(Transport Layer Security)를 사용합니다.
- API 서버에 액세스하는 데 X.509 인증서 또는 OAuth 액세스 토큰을 사용합니다.
- 프로젝트에 할당량을 지정하면 불량 토큰으로 인한 피해가 제한됩니다
- etcd 서비스는 클러스터에 직접 노출되지 않습니다

2.10.4. 플랫폼의 인증서 관리

OpenShift Container Platform의 프레임워크에는 TLS 인증서를 통한 암호화를 활용하는 REST 기반 HTTPS 통신을 사용하는 여러 구성요소가 있습니다. OpenShift Container Platform의 설치 관리자는 설치 중에 이러한 인증서를 구성합니다. 이 트래픽을 생성하는 몇 가지 기본 구성요소가 있습니다.

- 마스터(API 서버 및 컨트롤러)
- etcd
- 노드
- 레지스트리
- 라우터

2.10.4.1. 사용자 정의 인증서 구성

초기 설치 중 또는 인증서를 재배포할 때 API 서버와 웹 콘솔의 공개 호스트 이름에 맞게 사용자 정의 제공 인증서를 구성할 수 있습니다. 사용자 정의 CA도 사용할 수 있습니다.

추가 리소스

- OpenShift Container Platform 소개
- RBAC를 사용하여 권한 정의 및 적용
- 승인 플러그인 정보
- 보안 컨텍스트 제약 조건 관리
- SCC 참조 명령
- 서비스 계정에 역할을 부여하는 예
- 내부 OAuth 서버 구성
- ID 공급자 구성 이해
- 인증서 유형 및 설명
- 프록시 인증서

2.11. 네트워크 보안

네트워크 보안은 여러 수준에서 관리할 수 있습니다. Pod 수준에서 네트워크 네임스페이스는 네트워크 액세스를 제한하여 컨테이너에 다른 Pod 또는 호스트 시스템이 표시되지 않게 할 수 있습니다. 네트워크 정책을 통해 연결을 허용하거나 거부할 수 있습니다. 컨테이너화된 애플리케이션으로 수신 및 송신되는 트래픽을 관리할 수 있습니다.

2.11.1. 네트워크 네임스페이스 사용

OpenShift Container Platform에서는 소프트웨어 정의 네트워킹(SDN)을 사용하여 클러스터 전체의 컨테이너 간 통신이 가능한 통합 클러스터 네트워크를 제공합니다.

기본적으로 네트워크 정책 모드에서는 다른 Pod 및 네트워크 끝점에서 프로젝트의 모든 Pod에 액세스 할 수 있습니다. 프로젝트에서 하나 이상의 Pod를 분리하기 위해 해당 프로젝트에서 **NetworkPolicy** 오브젝트를 생성하여 수신되는 연결을 표시할 수 있습니다. 다중 테넌트 모드를 사용하면 Pod 및 서비스에 대한

프로젝트 수준 격리를 제공할 수 있습니다.

2.11.2. 네트워크 정책으로 Pod 분리

*네트워크 정책*을 사용하면 동일한 프로젝트에서 Pod를 서로 분리할 수 있습니다. 네트워크 정책은 Pod에 대한 모든 네트워크 액세스를 거부하고 수신 컨트롤러의 연결만 허용하거나 다른 프로젝트의 Pod에서 연결을 거부하거나 네트워크 작동 방식에 관한 유사한 규칙을 설정할 수 있습니다.

추가 리소스

- 네트워크 정책 정의

2.11.3. 여러 Pod 네트워크 사용

실행 중인 각 컨테이너에는 기본적으로 하나의 네트워크 인터페이스만 있습니다. Multus CNI 플러그인을 사용하면 여러 CNI 네트워크를 만든 다음 해당 네트워크를 Pod에 연결할 수 있습니다. 이렇게 하면 개인 데이터를 더 제한된 네트워크로 분리할 수 있으며 노드마다 네트워크 인터페이스가 여러 개일 수 있습니다.

추가 리소스

- 여러 네트워크 사용

2.11.4. 애플리케이션 격리

OpenShift Container Platform을 사용하면 단일 클러스터에서 네트워크 트래픽을 세그먼트화하여 사용자, 팀, 애플리케이션 및 환경을 글로벌이 아닌 리소스와 분리하는 다중 테넌트 클러스터를 만들 수 있습니다.

추가 리소스

- OpenShiftSDN을 사용하여 네트워크 격리 구성

2.11.5. 수신 트래픽 보안

OpenShift Container Platform 클러스터 외부에서 쿠버네티스 서비스에 대한 액세스를 구성하는 방법과 관련하여 보안에 미치는 영향이 많이 있습니다. HTTP 및 HTTPS 경로를 노출하는 외에도 수신 라우팅을 사용하면 NodePort 또는 LoadBalancer 수신 유형을 설정할 수 있습니다. NodePort에서는 각 클러스터 작업자의 애플리케이션 서비스 API 오브젝트를 노출합니다. LoadBalancer를 사용하면 OpenShift Container Platform 클러스터의 관련 서비스 API 오브젝트에 외부 로드 밸런서를 할당할 수 있습니다.

추가 리소스

- 수신 클러스터 트래픽 구성

2.11.6. 송신 트래픽 보안

OpenShift Container Platform에서는 라우터 또는 방화벽 방법을 사용하여 송신 트래픽을 제어하는 기능을 제공합니다. 예를 들어 IP 화이트리스트를 사용하여 데이터베이스 액세스를 제어할 수 있습니다. 클러스터 관리자는 OpenShift Container Platform SDN 네트워크 공급자의 프로젝트에 하나 이상의 송신 IP 주소를 할당할 수 있습니다. 마찬가지로 클러스터 관리자는 송신 방화벽을 사용하여 송신 트래픽이 OpenShift Container Platform 클러스터 외부로 나가는 것을 방지할 수 있습니다.

고정 송신 IP 주소를 할당하면 특정 프로젝트용으로 나가는 모든 트래픽을 해당 IP 주소에 할당할 수 있습니다. 송신 방화벽을 사용하면 Pod가 외부 네트워크에 연결 또는 Pod가 내부 네트워크에 연결하는 것을 방지하거나 특정 내부 서브넷에 대한 Pod의 액세스를 제한할 수 있습니다.

추가 리소스

- 외부 IP 주소에 대한 액세스를 제어하도록 송신 방화벽 구성
- 프로젝트의 송신 IP 구성

2.12. 연결된 스토리지 보안

OpenShift Container Platform에서는 온프레미스 및 클라우드 공급자를 위해 여러 유형의 스토리지를 지원합니다. 특히 OpenShift Container Platform에서는 컨테이너 스토리지 인터페이스를 지원하는 스토리지 유형을 사용할 수 있습니다.

2.12.1. 영구 볼륨 플러그인

컨테이너는 스테이트리스(stateless) 및 스테이트풀(stateful) 애플리케이션 둘 다에 유용합니다. 연결된 스토리지를 보호하는 것이 상태 저장 서비스 보안의 핵심 요소입니다. CSI(Container Storage Interface)를 사용하여 OpenShift Container Platform에서는 CSI 인터페이스를 지원하는 모든 스토리지 백엔드의 스토리지를 통합할 수 있습니다.

OpenShift Container Platform에서는 다음을 포함하여 여러 유형의 스토리지에 맞는 플러그인을 제공합니다.

- Red Hat OpenShift 컨테이너 스토리지*
- AWS EBS(Elastic Block Stores)*
- AWS EFS(Elastic File System)*
- Azure 디스크*
- Azure 파일*
- OpenStack Cinder*
- GCE 영구 디스크*
- VMware vSphere*
- 네트워크 파일 시스템(NFS)
- FlexVolume
- 파이버 채널
- iSCSI

동적 프로비저닝을 사용하는 스토리지 유형의 플러그인에는 별표(*)가 표시됩니다. 서로 통신 중인 모든 OpenShift Container Platform 구성요소에서 전송 중인 데이터는 HTTPS를 통해 암호화됩니다.

스토리지 유형에서 지원하는 방식으로 호스트에 영구 볼륨(PV)을 마운트할 수 있습니다. 스토리지 유형에 따라 기능이 다르며 각 PV의 액세스 모드는 해당 볼륨에서 지원하는 특정 모드로 설정됩니다.

예를 들어 NFS에서는 여러 읽기/쓰기 클라이언트를 지원할 수 있지만 특정 NFS PV는 서버에서 읽기 전용으로 내보낼 수 있습니다. 각 PV에는 **ReadWriteOnce**, **ReadOnlyMany** 및 **ReadWriteMany**와 같은 특정 PV 기능을 설명하는 자체 액세스 모드 세트가 있습니다.

2.12.2. 공유 스토리지

NFS와 같은 공유 스토리지 공급자의 경우 PV는 그룹 ID(GID)를 PV 리소스에 대한 주석으로 등록합니다. 그런 다음 Pod에서 PV를 요청하면 주석이 달린 GID가 Pod의 보충 그룹에 추가되므로, 해당 Pod가 공유 스토리지의 콘텐츠에 액세스할 수 있습니다.

2.12.3. 블록 스토리지

AWS EBS(Elastic Block Store), GCE 영구 디스크 및 iSCSI와 같은 블록 스토리지 공급자의 경우 OpenShift Container Platform에서는 SELinux 기능을 사용하여 권한이 없는 Pod용으로 마운트된 볼륨의 루트를 보호함으로써, 연관된 컨테이너에서만 마운트된 볼륨을 소유하고 볼 수 있게 합니다.

추가 리소스

- 영구저장장치 이해
- CSI 볼륨 구성
- 동적 프로비저닝
- NFS를 사용하는 영구저장장치
- AWS Elastic Block Store를 사용하는 영구저장장치
- GCE 영구 디스크를 사용한 영구저장장치

2.13. 클러스터 이벤트 및 로그 모니터링

OpenShift Container Platform 클러스터를 모니터링하고 감사하는 기능은 부적절한 사용으로부터 클러스터와 사용자를 보호하는 데 중요한 부분입니다.

이 용도에 유용한 클러스터 수준 정보의 두 가지 주요 소스, 즉 이벤트와 로깅이 있습니다.

2.13.1. 클러스터 이벤트 감시

클러스터 관리자는 **Event** 리소스 유형을 익히고 시스템 이벤트 목록을 검토하여 관심있는 이벤트를 결정하는 것이 좋습니다. 이벤트는 네임스페이스, 즉 관련 리소스의 네임스페이스 또는 클러스터 이벤트의 경우에는 **default** 네임스페이스와 연결됩니다. 기본 네임스페이스에는 인프라 구성요소와 관련된 노드 이벤트 및 리소스 이벤트와 같은 클러스터 모니터링 또는 감사 관련 이벤트가 있습니다.

마스터 API 및 **oc** 명령은 이벤트 목록의 범위를 노드 관련 항목으로만 지정하는 매개변수는 제공하지 않습니다. 간단한 접근 방식은 **grep**을 사용하는 것입니다.

```
$ oc get event -n default | grep Node
```

출력 예

```
1h         20h         3         origin-node-1.example.local   Node      Normal    NodeHasDiskPressure   ...
```

더 유연한 접근 방식은 다른 툴에서 처리할 수 있는 양식으로 이벤트를 출력하는 것입니다. 예를 들어 다음 예에서는 JSON 출력을 대상으로 **jq** 툴을 사용하여 **NodeHasDiskPressure** 이벤트만 추출합니다.

```
$ oc get events -n default -o json \
  | jq '.items[] | select(.involvedObject.kind == "Node" and .reason == "NodeHasDiskPressure")'
```

출력 예

```
{
  "apiVersion": "v1",
  "count": 3,
  "involvedObject": {
    "kind": "Node",
    "name": "origin-node-1.example.local",
    "uid": "origin-node-1.example.local"
  },
  "kind": "Event",
  "reason": "NodeHasDiskPressure",
  ...
}
```

리소스 생성, 수정 또는 삭제와 관련된 이벤트도 클러스터의 오용을 탐지하는 데 적합할 수 있습니다. 예를 들어 다음과 같은 쿼리를 사용하면 과도한 이미지 가져오기를 찾을 수 있습니다.

```
$ oc get events --all-namespaces -o json \
  | jq '[.items[] | select(.involvedObject.kind == "Pod" and .reason == "Pulling")] | length'
```

출력 예

```
4
```

참고

네임스페이스가 삭제되면 해당 이벤트도 삭제됩니다. 이벤트도 만료될 수 있으며 etcd 스토리지가 가득 차지 않도록 삭제됩니다. 이벤트는 영구 레코드로 저장되지 않으며 시간 경과에 따른 통계를 캡처하려면 자주 폴링해야 합니다.

2.13.2. 로깅

oc log 명령을 사용하면 컨테이너 로그, 빌드 구성, 배포를 실시간으로 볼 수 있습니다. 다른 사용자는 다른 로그에 액세스할 수 있습니다.

- 프로젝트에 액세스할 수 있는 사용자는 기본적으로 해당 프로젝트의 로그를 볼 수 있습니다.
- 관리자 역할이 있는 사용자는 모든 컨테이너 로그에 액세스할 수 있습니다.

추가 감사 및 분석을 위해 로그를 저장하려면 **cluster-logging** 애드온 기능을 사용하여 시스템, 컨테이너 및 감사 로그를 수집, 관리 및 볼 수 있습니다. OpenShift Elasticsearch Operator 및 Cluster Logging Operator를 통해 클러스터 로깅을 배포, 관리 및 업그레이드할 수 있습니다.

2.13.3. 감사 로그

*감사 로그*를 사용하면 사용자, 관리자 또는 기타 OpenShift Container Platform 구성요소의 동작과 관련된 일련의 활동을 따를 수 있습니다. API 감사 로깅은 각 서버에서 수행됩니다.

추가 리소스

- 시스템 이벤트 목록
- 클러스터 로깅 이해
- 감사 로그 보기

3장. 인증서 구성

3.1. 기본 수신 인증서 교체

3.1.1. 기본 수신 인증서 이해

기본적으로 OpenShift Container Platform에서는 Ingress Operator를 사용하여 내부 CA를 생성하고 **.apps** 하위 도메인의 애플리케이션에 유효한 와일드카드 인증서를 발급합니다. 웹 콘솔과 CLI도 모두 이 인증서를 사용합니다.

내부 인프라 CA 인증서는 자체 서명됩니다. 이 프로세스는 일부 보안 또는 PKI 팀에서는 나쁜 습관으로 인식할 수 있지만 위험이 거의 없습니다. 이러한 인증서를 암시적으로 신뢰하는 유일한 클라이언트는 클러스터 내의 다른 구성요소입니다. 기본 와일드카드 인증서를 컨테이너 사용자 공간에서 제공한 대로 이미 CA 번들에 포함된 공용 CA에서 발행한 인증서로 교체하면 외부 클라이언트가 **.apps** 하위 도메인에서 실행되는 애플리케이션에 안전하게 연결할 수 있습니다.

3.1.2. 기본 수신 인증서 교체

.apps 하위 도메인에 있는 애플리케이션의 기본 수신 인증서를 교체할 수 있습니다. 인증서를 교체하고 나면 웹 콘솔 및 CLI를 포함한 모든 애플리케이션에는 지정된 인증서에서 제공하는 암호화가 생깁니다.

사전 요구 사항

- 정규화된 **.apps** 하위 도메인 및 해당 개인 키에 맞는 와일드카드 인증서가 있어야 합니다. 각각은 별도의 PEM 형식 파일이어야 합니다.
- 개인 키는 암호화되지 않아야 합니다. 키가 암호화된 경우 OpenShift Container Platform으로 가져오기 전에 키의 암호를 해독합니다.
- 인증서에는 ***.apps.<clustername>.<domain>**을 표시하는 **subjectAltName** 확장자가 포함되어야 합니다.
- 인증서 파일은 체인에 하나 이상의 인증서를 포함할 수 있습니다. 와일드카드 인증서는 파일에서 첫 번째 인증서여야 합니다. 그런 다음 중간 인증서가 올 수 있으며 파일은 루트 CA 인증서로 끝나야 합니다.
- 루트 CA 인증서를 추가 PEM 형식 파일로 복사합니다.

프로세스

1. 와일드카드 인증서에 서명하는 데 사용된 루트 CA 인증서만 포함하는 구성 맵을 생성합니다.

```
$ oc create configmap custom-ca \
    --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
    -n openshift-config
```

1 **</path/to/example-ca.crt>**는 로컬 파일 시스템에서 루트 CA 인증서 파일의 경로입니다.

2. 새로 생성된 구성 맵으로 클러스터 전체 프록시 구성을 업데이트합니다.

```
$ oc patch proxy/cluster \
    --type=merge \
    --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'
```

3. 와일드카드 인증서 체인과 키를 포함하는 보안을 생성합니다.

```
$ oc create secret tls <secret> \ 1
    --cert=</path/to/cert.crt> \ 2
    --key=</path/to/cert.key> \ 3
    -n openshift-ingress
```

1 **<secret>**은 인증서 체인과 개인 키를 포함할 보안의 이름입니다.

2 **</path/to/cert.crt>**는 로컬 파일 시스템의 인증서 체인 경로입니다.

3 **</path/to/cert.key>**는 이 인증서와 관련된 개인 키의 경로입니다.

4. 새로 생성된 보안으로 Ingress Controller 구성을 업데이트합니다.

```
$ oc patch ingresscontroller.operator default \
    --type=merge -p \
    '{"spec":{"defaultCertificate": {"name": "<secret>"}}}' \ 1
    -n openshift-ingress-operator
```

1 **<secret>**을 이전 단계에서 보안에 사용된 이름으로 교체합니다.

추가 리소스

- CA 번들 인증서 교체
- 프록시 인증서 사용자 정의

3.2. API 서버 인증서 추가

기본 API 서버 인증서는 내부 OpenShift Container Platform 클러스터 CA에서 발급합니다. 클러스터 외부의 클라이언트는 기본적으로 API 서버의 인증서를 확인할 수 없습니다. 이 인증서는 클라이언트가 신뢰하는 CA에서 발급한 인증서로 교체할 수 있습니다.

3.2.1. certificate이라는 API 서버 추가

기본 API 서버 인증서는 내부 OpenShift Container Platform 클러스터 CA에서 발급합니다. 리버스 프록시 또는 로드 밸런서가 사용되는 경우와 같이 클라이언트가 요청한 정규화된 도메인 이름(FQDN)을 기반으로 API 서버가 반환할 대체 인증서를 하나 이상 추가할 수 있습니다.

사전 요구 사항

- FQDN의 인증서와 해당 개인 키가 있어야 합니다. 각각은 별도의 PEM 형식 파일이어야 합니다.
- 개인 키는 암호화되지 않아야 합니다. 키가 암호화된 경우 OpenShift Container Platform으로 가져오기 전에 키의 암호를 해독합니다.
- 인증서에는 FQDN을 표시하는 **subjectAltName** 확장자가 포함되어야 합니다.
- 인증서 파일은 체인에 하나 이상의 인증서를 포함할 수 있습니다. API 서버 FQDN의 인증서는 파일의 첫 번째 인증서여야 합니다. 그런 다음 중간 인증서가 올 수 있으며 파일은 루트 CA 인증서로 끝나야 합니다.

주의

내부 로드 밸런서용으로 이름 지정된 인증서를 제공하지 마십시오(host name **api-int.<cluster_name>.<base_domain>**). 그렇지 않으면 클러스터 성능이 저하됩니다.

프로세스

1. **kubeadmin** 사용자로 새 API에 로그인합니다.

```
$ oc login -u kubeadmin -p <password> https://FQDN:6443
```

2. **kubeconfig** 파일을 가져옵니다.

```
$ oc config view --flatten > kubeconfig-newapi
```

3. **openshift-config** 네임스페이스에 인증서 체인과 개인 키가 포함된 보안을 생성합니다.

```
$ oc create secret tls <secret> \ 1
    --cert=</path/to/cert.crt> \ 2
    --key=</path/to/cert.key> \ 3
    -n openshift-config
```

1. **<secret>**은 인증서 체인과 개인 키를 포함할 보안의 이름입니다.
2. **</path/to/cert.crt>**는 로컬 파일 시스템의 인증서 체인 경로입니다.
3. **</path/to/cert.key>**는 이 인증서와 관련된 개인 키의 경로입니다.

4. 생성된 보안을 참조하도록 API 서버를 업데이트합니다.

```
$ oc patch apiserver cluster \
    --type=merge -p \
    '{"spec":{"servingCerts": {"namedCertificates":
    [{"names": ["<FQDN>"], 1
    "servingCertificate": {"name": "<secret>"}}]}}' 2
```

1. **<FQDN>**을 API 서버가 인증서를 제공해야 하는 FQDN으로 바꿉니다.
2. **<secret>**을 이전 단계에서 보안에 사용된 이름으로 교체합니다.

5. **apiserver/cluster** 오브젝트를 조사하고 보안이 이제 참조되는지 확인합니다.

```
$ oc get apiserver cluster -o yaml
```

출력 예

```
...
```

```
spec:
  servingCerts:
    namedCertificates:
    - names:
      - <FQDN>
      servingCertificate:
        name: <secret>
...
```

6. **kube-apiserver** 운영자를 확인하고 Kubernetes API 서버의 새 버전이 롤아웃되는지 확인합니다. 운영자가 구성 변경 사항을 탐지하고 새 배포를 트리거하는 데 1분 정도 걸릴 수 있습니다. 새 버전이 롤아웃되는 동안 **PROGRESSING**은 **True**를 보고합니다.

```
$ oc get clusteroperators kube-apiserver
```

다음 출력에 표시된 것처럼 **PROGRESSING**이 **False**로 표시될 때까지 다음 단계를 진행하지 마십시오.

출력 예

```
NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
kube-apiserver   4.6.0     True        False         False      145m
```

PROGRESSING에 **True**가 표시되면 몇 분 기다렸다가 다시 시도하십시오.

3.3. 서비스 제공 인증서 보안을 사용하여 서비스 트래픽 보안

3.3.1. 서비스 제공 인증서 이해

서비스 제공 인증서는 암호화가 필요한 복잡한 미들웨어 애플리케이션을 지원하기 위한 것입니다. 이러한 인증서는 TLS 웹 서버 인증서로 발급됩니다.

service-ca 컨트롤러에서는 **x509.SHA256WithRSA** 서명 알고리즘을 사용하여 서비스 인증서를 생성합니다.

생성된 인증서 및 키는 PEM 형식이며, 생성된 보안의 **tls.crt** 및 **tls.key**에 각각 저장됩니다. 인증서와 키는 만료 시기가 다가오면 자동으로 교체됩니다.

서비스 인증서를 발급하는 서비스 CA 인증서는 26개월 동안 유효하며 유효 기간이 13개월 미만으로 남아 있으면 자동으로 순환됩니다. 교체 후에도 이전 서비스 CA 구성은 만료될 때까지 계속 신뢰 상태가 유지됩니다. 그러면 만료되기 전에 유예 기간 동안 영향을 받는 모든 서비스의 주요 자료를 새로 고칠 수 있습니다. 서비스를 다시 시작하고 키 자료를 새로 고치는 이 유예 기간 동안 클러스터를 업그레이드하지 않으면 이전 서비스 CA가 만료된 후 실패하지 않도록 서비스를 직접 다시 시작해야 할 수도 있습니다.

참고

다음 명령을 사용하여 클러스터의 모든 Pod를 직접 다시 시작할 수 있습니다. 이 명령을 실행하면 모든 네임스페이스에서 실행 중인 모든 Pod가 삭제되므로 서비스가 중단됩니다. 이 Pod는 삭제 후 자동으로 다시 시작됩니다.

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
    do oc delete pods --all -n $I; \
    sleep 1; \
    done
```

3.3.2. 서비스 인증서 추가

서비스와의 통신을 보호하려면 서명된 제공 인증서와 키 쌍을 서비스와 동일한 네임스페이스의 보안에 생성합니다.

중요

생성된 인증서는 내부 서비스 DNS 이름 **<service.name>.<service.namespace>.svc**에만 유효하고 내부 통신에만 유효합니다.

사전 요구 사항

- 서비스가 정의되어 있어야 합니다.

프로세스

1. **service.beta.openshift.io/serving-cert-secret-name**으로 서비스에 주석을 답니다.

```
$ oc annotate service <service_name> \ 1
    service.beta.openshift.io/serving-cert-secret-name=<secret_name> 2
```

1 **<service_name>**을 보호할 서비스 이름으로 교체합니다.

2 **<secret_name>**은 인증서와 키 쌍이 포함되어 생성된 보안의 이름입니다. **<service_name>**과 동일하게 설정하면 편리합니다.

예를 들어 서비스 **test1**에 주석을 달려면 다음 명령을 사용합니다.

```
$ oc annotate service test1 service.beta.openshift.io/serving-cert-secret-name=test1
```

2. 서비스를 검사하여 주석이 있는지 확인합니다.

```
$ oc describe service <service_name>
```

출력 예

```
...
Annotations:              service.beta.openshift.io/serving-cert-secret-name: <service_name>
                          service.beta.openshift.io/serving-cert-signed-by: openshift-service-serving-
signer@1556850837
...
```

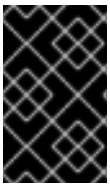
3. 클러스터에서 서비스 보안을 생성하면 **Pod** 사양에서 보안을 마운트하고, 사용 가능 상태가 되면 Pod에서 실행합니다.

추가 리소스

- 서비스 인증서를 사용하여 재암호화 TLS 종료를 사용하여 보안 경로를 구성할 수 있습니다. 자세한 내용은 사용자 정의 인증서를 사용하여 재암호화 경로 생성 을 참조하십시오.

3.3.3. 구성 맵에 서비스 CA 번들 추가

Pod는 **service.beta.openshift.io/inject-cabundle=true** 로 주석이 달린 **ConfigMap** 오브젝트를 마운트하여 서비스 CA 인증서에 액세스할 수 있습니다. 주석이 달리면 클러스터에서 서비스 CA 인증서를 구성 맵의 **service-ca.crt** 키에 자동으로 삽입합니다. 이 CA 인증서에 액세스하면 TLS 클라이언트가 서비스 제공 인증서를 사용하여 서비스에 대한 연결을 확인할 수 있습니다.



중요

구성 맵에 이 주석을 달면 기존 데이터가 모두 삭제됩니다. Pod 구성을 저장하는 것과 동일한 구성 맵을 사용하는 대신 별도의 구성 맵을 사용하여 **service-ca.crt**를 포함하는 것이 좋습니다.

프로세스

1. **service.beta.openshift.io/inject-cabundle=true**로 구성 맵에 주석을 답니다.

```
$ oc annotate configmap <config_map_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 **<config_map_name>**을 주석을 달 구성 맵 이름으로 교체합니다.

참고

볼륨 마운트 시 **service-ca.crt** 키를 명시적으로 참조하면 CA 번들로 구성 맵을 삽입할 때까지 Pod가 시작되지 않습니다. 이 동작은 볼륨 제공 인증서 구성에서 **optional** 필드를 **true**로 설정하여 덮어쓸 수 있습니다.

예를 들어 구성 맵 **test1**에 주석을 달려면 다음 명령을 사용합니다.

```
$ oc annotate configmap test1 service.beta.openshift.io/inject-cabundle=true
```

2. 구성 맵을 보고 서비스 CA 번들이 삽입되었는지 확인합니다.

```
$ oc get configmap <config_map_name> -o yaml
```

CA 번들은 YAML 출력에서 **service-ca.crt** 키 값으로 표시됩니다.

```
apiVersion: v1
data:
  service-ca.crt: |
    -----BEGIN CERTIFICATE-----
...
```

3.3.4. API 서비스에 서비스 CA 번들 추가

spec.caBundle 필드에 서비스 CA 번들이 입력되도록 **service.beta.openshift.io/inject-cabundle=true**로 **APIService** 오브젝트에 주석을 답니다. 그러면 쿠버네티스 API 서버가 대상 끝점을 보호하는 데 사용되는 서비스 CA 인증서의 유효성을 확인할 수 있습니다.

프로세스

1. **service.beta.openshift.io/inject-cabundle=true**로 API 서비스에 주석을 답니다.

```
$ oc annotate apiservice <api_service_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 **<api_service_name>**을 주석을 달 API 서비스 이름으로 교체합니다.

예를 들어 API 서비스 **test1**에 주석을 달려면 다음 명령을 사용합니다.

```
$ oc annotate apiservice test1 service.beta.openshift.io/inject-cabundle=true
```

2. API 서비스를 보고 서비스 CA 번들이 삽입되었는지 확인합니다.

```
$ oc get apiservice <api_service_name> -o yaml
```

CA 번들은 YAML 출력의 **spec.caBundle** 필드에 표시됩니다.

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
spec:
  caBundle: <CA_BUNDLE>
...
```

3.3.5. 사용자 정의 리소스 정의에 서비스 CA 번들 추가

spec.conversion.webhook.clientConfig.caBundle 필드에 서비스 CA 번들이 입력되도록 **CustomResourceDefinition**(CRD) 오브젝트에 **service.beta.openshift.io/inject-cabundle=true**로 주석을 답니다. 그러면 쿠버네티스 API 서버가 대상 끝점을 보호하는 데 사용되는 서비스 CA 인증서의 유효성을 확인할 수 있습니다.

참고

CRD가 변환에 웹 후크를 사용하도록 구성된 경우 서비스 CA 번들은 CRD에만 삽입됩니다. CRD의 웹 후크가 서비스 CA 인증서로 보안된 경우에만 서비스 CA 번들을 삽입하는 것이 유용합니다.

프로세스

1. **service.beta.openshift.io/inject-cabundle=true**로 CRD에 주석을 답니다.

```
$ oc annotate crd <crd_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 **<crd_name>**을 주석을 달 CRD 이름으로 교체합니다.

예를 들어 CRD **test1**에 주석을 달려면 다음 명령을 사용합니다.

```
$ oc annotate crd test1 service.beta.openshift.io/inject-cabundle=true
```

2. CRD를 보고 서비스 CA 번들이 삽입되었는지 확인합니다.

```
$ oc get crd <crd_name> -o yaml
```

CA 번들은 YAML 출력의 **spec.conversion.webhook.clientConfig.caBundle** 필드에 표시됩니다.

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
spec:
  conversion:
    strategy: Webhook
    webhook:
      clientConfig:
        caBundle: <CA_BUNDLE>
...
```

3.3.6. 변경 웹 후크 구성에 서비스 CA 번들 추가

각 웹 후크의 **clientConfig.caBundle** 필드에 서비스 CA 번들이 입력되도록 **service.beta.openshift.io/inject-cabundle=true**로 **MutatingWebhookConfiguration** 오브젝트에 주석을 달 수 있습니다. 그러면 쿠버네티스 API 서버가 대상 끝점을 보호하는 데 사용되는 서비스 CA 인증서의 유효성을 확인할 수 있습니다.

참고

웹 후크마다 다른 CA 번들을 지정해야 하는 승인 웹 후크 구성에는 이 주석을 설정하지 마십시오. 그러면 모든 웹 후크에 서비스 CA 번들이 삽입됩니다.

프로세스

1. **service.beta.openshift.io/inject-cabundle=true**로 변경 웹 후크 구성에 주석을 답니다.

```
$ oc annotate mutatingwebhookconfigurations <mutating_webhook_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 **<mutating_webhook_name>**을 주석을 달 변경 웹 후크 구성 이름으로 교체합니다.

예를 들어 변경 웹 후크 구성 **test1**에 주석을 달려면 다음 명령을 사용합니다.

```
$ oc annotate mutatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. 변경 웹 후크 구성을 보고 서비스 CA 번들이 삽입되었는지 확인합니다.

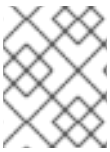
```
$ oc get mutatingwebhookconfigurations <mutating_webhook_name> -o yaml
```

CA 번들은 YAML 출력에 있는 모든 웹 후크의 **clientConfig.caBundle** 필드에 표시됩니다.

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

3.3.7. 검증 웹 후크 구성에 서비스 CA 번들 추가

각 웹 후크의 **clientConfig.caBundle** 필드에 서비스 CA 번들이 입력되도록 **service.beta.openshift.io/inject-cabundle=true**로 **ValidatingWebhookConfiguration** 오브젝트에 주석을 달 수 있습니다. 그러면 쿠버네티스 API 서버가 대상 끝점을 보호하는 데 사용되는 서비스 CA 인증서의 유효성을 확인할 수 있습니다.



참고

웹 후크마다 다른 CA 번들을 지정해야 하는 승인 웹 후크 구성에는 이 주석을 설정하지 마십시오. 그러면 모든 웹 후크에 서비스 CA 번들이 삽입됩니다.

프로세스

1. **service.beta.openshift.io/inject-cabundle=true**로 검증 웹 후크 구성에 주석을 답니다.

```
$ oc annotate validatingwebhookconfigurations <validating_webhook_name> \ (1)
    service.beta.openshift.io/inject-cabundle=true
```

(1) **<validating_webhook_name>**을 주석을 달 검증 웹 후크 구성 이름으로 교체합니다.

예를 들어 검증 웹 후크 구성 **test1**에 주석을 달려면 다음 명령을 사용합니다.

```
$ oc annotate validatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. 검증 웹 후크 구성을 보고 서비스 CA 번들이 삽입되었는지 확인합니다.

```
$ oc get validatingwebhookconfigurations <validating_webhook_name> -o yaml
```

CA 번들은 YAML 출력에 있는 모든 웹 후크의 **clientConfig.caBundle** 필드에 표시됩니다.

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

3.3.8. 생성된 서비스 인증서를 직접 순환

관련 보안을 삭제하여 서비스 인증서를 순환할 수 있습니다. 보안을 삭제하면 새로운 보안이 자동으로 생성되어 새 인증서가 생성됩니다.

사전 요구 사항

- 인증서 및 키 쌍을 포함하는 보안이 서비스용으로 생성되어야 합니다.

프로세스

1. 서비스를 검사하여 인증서가 포함된 보안을 판별합니다. 아래 표시된 대로 **serving-cert-secret-name** 주석에 있습니다.

```
$ oc describe service <service_name>
```

출력 예

```
...
service.beta.openshift.io/serving-cert-secret-name: <secret>
...
```

2. 서비스용으로 생성된 보안을 삭제합니다. 이 프로세스는 자동으로 보안을 재생성합니다.

```
$ oc delete secret <secret> 1
```

1 **<secret>**을 이전 단계의 보안 이름으로 교체합니다.

3. 새 보안을 확보하고 **AGE**를 검사하여 인증서가 다시 생성되었는지 확인합니다.

```
$ oc get secret <service_name>
```

출력 예

```
NAME              TYPE                DATA   AGE
<service.name>    kubernetes.io/tls   2      1s
```

3.3.9. 서비스 CA 인증서를 직접 순환

서비스 CA는 26개월 동안 유효하며 유효 기간이 13개월 미만으로 남아 있으면 자동으로 새로 고쳐집니다.

필요하면 다음 절차에 따라 서비스 CA를 직접 새로 고칠 수 있습니다.

주의

수동으로 순환된 서비스 CA는 이전 서비스 CA와의 신뢰를 유지 관리하지 않습니다. 클러스터의 Pod가 다시 시작되어 Pod가 새 서비스 CA에서 발급한 서비스 제공 인증서를 사용하고 있는지 확인할 때까지 일시적으로 서비스 중단이 발생할 수 있습니다.

사전 요구 사항

- 클러스터 관리자로 로그인해야 합니다.

프로세스

1. 다음 명령을 사용하여 현재 서비스 CA 인증서의 만료 날짜를 봅니다.

```
$ oc get secrets/signing-key -n openshift-service-ca \
    -o template='{{index .data "tls.crt"}}' \
    | base64 --decode \
    | openssl x509 -noout -enddate
```

2. 서비스 CA를 직접 순환합니다. 이 프로세스는 새로운 서비스 인증서에 서명하는 데 사용될 새로운 서비스 CA를 생성합니다.

```
$ oc delete secret/signing-key -n openshift-service-ca
```

3. 새 인증서를 모든 서비스에 적용하도록 클러스터의 모든 Pod를 다시 시작합니다. 이 명령을 실행하면 모든 서비스가 업데이트된 인증서를 사용합니다.

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
    do oc delete pods --all -n $I; \
    sleep 1; \
    done
```

주의

이 명령은 모든 네임스페이스에서 실행 중인 모든 Pod를 대상으로 진행하고 삭제하므로 서비스가 중단됩니다. 이 Pod는 삭제 후 자동으로 다시 시작됩니다.

3.4. CA 번들 업데이트

3.4.1. CA 번들 인증서 이해

프록시 인증서를 사용하면 송신 연결을 만들 때 플랫폼 구성 요소에서 사용하는 하나 이상의 사용자 정의 인증 기관(CA)을 지정할 수 있습니다.

Proxy 오브젝트의 **trustedCA** 필드는 사용자 제공 신뢰할 수 있는 인증 기관(CA) 번들을 포함하는 구성 맵에 대한 참조입니다. 이 번들은 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들과 병합되어 송신 HTTPS 호출을 수행하는 플랫폼 구성요소의 신뢰 저장소에 삽입됩니다. 예를 들어 **image-registry-operator**에서 이미지를 다운로드하도록 외부 이미지 레지스트리를 호출합니다. **trustedCA**를 지정하지 않으면 프록시 HTTPS 연결에 RHCOS 신뢰 번들만 사용됩니다. 자체 인증서 인프라를 사용하려면 RHCOS 신뢰 번들에 사용자 정의 CA 인증서를 제공합니다.

trustedCA 필드는 프록시 유효성 검증기만 사용해야 합니다. 유효성 검증기를 통해서는 필수 키 **ca-bundle.crt**에서 인증서 번들을 읽고 **openshift-config-managed** 네임스페이스의 **trusted-ca-bundle**이라는 구성 맵에 복사해야 합니다. **trustedCA**에서 참조하는 구성 맵의 네임스페이스는 **openshift-config**입니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

3.4.2. CA 번들 인증서 교체

프로세스

1. 와일드카드 인증서에 서명하는 데 사용된 루트 CA 인증서가 포함된 구성 맵을 생성합니다.

```
$ oc create configmap custom-ca \
    --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
    -n openshift-config
```

1 **</path/to/example-ca.crt>**는 로컬 파일 시스템의 CA 인증서 번들 경로입니다.

2. 새로 생성된 구성 맵으로 클러스터 전체 프록시 구성을 업데이트합니다.

```
$ oc patch proxy/cluster \
    --type=merge \
    --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'
```

추가 리소스

- 기본 수신 인증서 교체
- 클러스터 전체 프록시 사용
- 프록시 인증서 사용자 정의

4장. 인증서 유형 및 설명

4.1. API 서버의 사용자 제공 인증서

4.1.1. 목적

API 서버는 **api.<cluster_name>.<base_domain>**에서 클러스터 외부의 클라이언트가 액세스할 수 있습니다. 클라이언트가 다른 호스트 이름으로 또는 클러스터 관리 인증 기관(CA) 인증서를 클라이언트에 배포하지 않고 API 서버에 액세스하게 합니다. 관리자는 콘텐츠를 제공할 때 API 서버가 사용할 사용자 정의 기본 인증서를 설정해야 합니다.

4.1.2. 위치

사용자 제공 인증서는 **openshift-config** 네임스페이스에 **kubernetes.io/tls** 유형 **Secret**으로 제공되어야 합니다. API 서버 클러스터 구성인 **apiserver/cluster** 리소스를 업데이트하여 사용자 제공 인증서를 사용할 수 있게 합니다.

4.1.3. 관리

사용자 제공 인증서는 사용자가 관리합니다.

4.1.4. 만료

API 서버 클라이언트 인증서가 5분 내에 만료됩니다.

사용자 제공 인증서는 사용자가 관리합니다.

4.1.5. 사용자 정의

필요에 따라 사용자 관리 인증서가 포함된 보안을 업데이트합니다.

추가 리소스

- API 서버 인증서 추가

4.2. 프록시 인증서

4.2.1. 목적

프록시 인증서를 사용하면 송신 연결을 만들 때 플랫폼 구성요소에서 사용하는 하나 이상의 사용자 정의 인증 기관(CA) 인증서를 지정할 수 있습니다.

Proxy 오브젝트의 **trustedCA** 필드는 사용자 제공 신뢰할 수 있는 인증 기관(CA) 번들을 포함하는 구성 맵에 대한 참조입니다. 이 번들은 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들과 병합되어 송신 HTTPS 호출을 수행하는 플랫폼 구성요소의 신뢰 저장소에 삽입됩니다. 예를 들어 **image-registry-operator**에서 이미지를 다운로드하도록 외부 이미지 레지스트리를 호출합니다. **trustedCA**를 지정하지 않으면 프록시 HTTPS 연결에 RHCOS 신뢰 번들만 사용됩니다. 자체 인증서 인프라를 사용하려면 RHCOS 신뢰 번들에 사용자 정의 CA 인증서를 제공합니다.

trustedCA 필드는 프록시 유효성 검증기만 사용해야 합니다. 유효성 검증기를 통해서는 필수 키 **ca-bundle.crt**에서 인증서 번들을 읽고 **openshift-config-managed** 네임스페이스의 **trusted-ca-bundle**이라는 구성 맵에 복사해야 합니다. **trustedCA**에서 참조하는 구성 맵의 네임스페이스는 **openshift-**

config입니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

추가 리소스

- 클러스터 전체 프록시 구성

4.2.2. 설치 중 프록시 인증서 관리

설치 프로그램 구성의 **additionalTrustBundle** 값은 설치 중 프록시 신뢰 CA 인증서를 지정하는 데 사용됩니다. 예를 들면 다음과 같습니다.

```
$ cat install-config.yaml
```

출력 예

```
...
proxy:
  httpProxy: http://<https://username:password@proxy.example.com:123/>
  httpsProxy: https://<https://username:password@proxy.example.com:123/>
 noProxy: <123.example.com,10.88.0.0/16>
additionalTrustBundle: |
    -----BEGIN CERTIFICATE-----
   <MY_HTTPS_PROXY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...
```

4.2.3. 위치

사용자 제공 신뢰 번들은 구성 맵으로 표시됩니다. 구성 맵은 송신 HTTPS 호출을 수행하는 플랫폼 구성 요소의 파일 시스템에 마운트됩니다. 일반적으로 Operator는 구성 맵을 **/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**에 마운트하지만 프록시에는 필요하지 않습니다. 프록시는 HTTPS 연결을 수정하거나 검사할 수 있습니다. 두 경우 모두 프록시는 연결을 위해 새 인증서를 생성하고 서명해야 합니다.

완전한 프록시 지원이란 지정된 프록시에 연결하고 생성된 서명을 신뢰한다는 의미입니다. 따라서 사용자가 신뢰할 수 있는 루트를 지정하고 이 신뢰할 수 있는 루트에 연결된 인증서 체인도 신뢰할 수 있게 해야 합니다.

RHCOS 신뢰 번들을 사용하는 경우 **/etc/pki/ca-trust/source/anchors**에 CA 인증서를 둡니다.

자세한 정보는 Red Hat Enterprise Linux 설명서에서 공유 시스템 인증서 사용을 참조하십시오.

4.2.4. 만료

사용자가 사용자 제공 신뢰 번들의 만료 기간을 설정합니다.

기본 만료 기간은 CA 인증서 자체를 통해 정의됩니다. OpenShift Container Platform 또는 RHCOS에서 사용하기 전에 인증서에 맞게 기본 만료 기간을 구성하는 것은 CA 관리자의 책임입니다.

참고

Red Hat에서는 CA 만료 시점을 모니터링하지 않습니다. 그러나 CA의 수명이 길기 때문에 일반적으로 문제가 되지 않습니다. 그러나 신뢰 번들을 정기적으로 업데이트해야 할 수도 있습니다.

4.2.5. 서비스

기본적으로 송신 HTTPS 호출을 수행하는 모든 플랫폼 구성요소에서는 RHCOS 신뢰 번들을 사용합니다. **trustedCA**가 정의된 경우에도 사용됩니다.

RHCOS 노드에서 실행 중인 모든 서비스는 노드의 신뢰 번들을 사용할 수 있습니다.

4.2.6. 관리

이러한 인증서는 사용자가 아닌 시스템에서 관리합니다.

4.2.7. 사용자 정의

사용자 제공 신뢰 번들 업데이트는 다음 중 하나로 구성됩니다.

- **trustedCA**에서 참조하는 구성 맵에서 PEM 인코딩 인증서 업데이트
- 새 신뢰 번들이 포함된 네임스페이스 **openshift-config**에 구성 맵을 생성하고 새 구성 맵 이름을 참조하도록 **trustedCA** 업데이트

RHCOS 신뢰 번들에 CA 인증서를 작성하는 메커니즘은 다른 파일을 RHCOS에 작성하는 방법과 같습니다. 이 방법은 머신 구성을 사용하여 수행됩니다. MCO(Machine Config Operator)에서 새 CA 인증서가 포함된 새로운 머신 구성을 적용하면 노드가 재부팅됩니다. 다음 번 부팅 중에 **coreos-update-ca-trust.service** 서비스는 RHCOS 노드에서 실행되며 새 CA 인증서를 사용하여 신뢰 번들을 자동으로 업데이트합니다. 예를 들면 다음과 같습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-examplecorp-ca-cert
spec:
  config:
    ignition:
      version: 3.1.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVORENDQXh5Z0F3SUJBZ0lKQU5
1bkkwRDY2MmNuTUEwR0NTcUdTSWIzRFFFQkN3VUFNSUdsTVFzd0NRWUQKV1FRR0V3SlZVek
VYTUJVR0ExVUVDQXdPVG05eWRHZ2dRMkZ5YjJ4cGJtRXhFREFPQmdOVkJBY01CMUpoYkdWcA
pBMmd4RmpBVUJnTlZCQW9NRFZKbFpDQklZWFFzSUVsdVl5NHhFekFSQmdOVkJBc01DbEpsWk
```

```
NCSVlYUWdTVlF4Ckh6QVpCZ05WQkFNTUVsSmxaQ0JJWVhRZ1NWUWdVbTl2ZENCRFFURWhN
QjhHQ1NxR1NJYjNEUUVKQVJZU2FXNW0KWGpDQnBURUxNQWtHQTFVRUJoTUNWVk14RnpBV
kJnTlZCQWdNRGs1dmNuUm9JRU5oY205c2FXNWhNUkF3RGdZRApXUVFIREFkU1lXeGxhV2RvTV
JZd0ZBWURWUVFLREExU1pXUWdTR0YwTENCSmJtTXVNUk13RVFZRFZRUUxEQXBTCkFXUWd
TR0YwSUVsVU1Sc3dHUVlEVlFRRERCSlNaV1FnU0dGMElFbFVJRkp2YjNRZ1EwRXhJVEFmQmdrc
WhraUcKMHcwQkNRRVdFbWx1Wm05elpXTkFjbVZrYUdGMExtTnZiVENDQVNJd0RRWUpLb1pJaH
ZjTkFRRUJCUUFEZ2dFUApCRENDQVFvQ2dnRUJBTFF0OU9KUWg2R0M1TFQxZzgwcU5oMHU1
MEJRNHNaL3laOGFFVHh0KzVsblBWWDZNSEt6CmQvaTdsRHFUZlRjZkxMMm55VUJkMmZRRGsx
QjBmeHJza2hHSUlaM2lmUDFQczRsdFRrdjhoUlNvYjNWdE5xU28KSHhrS2Z2RDJQS2pUUHhEUFdZ
eXJ1eTlpckxaaW9NZmZpM2kvZ0N1dDBaV3RBeU8zTVZlNXFXRi9lbkt3Z1BFUwpZOXBvK1RkQ3ZS
Qi9SVU9iQmFNNzYxRWNyTFNNMUdxSE51ZVNmcW5obzNBakxRNmRCblBXbG82MzhabTFWZWJ
LCkNFTHloa0xXTVNGa0t3RG1uZTBqUTAyWTRnMDc1dkNLdkNzQ0F3RUFBYU5qTUdFd0hRWUR
WUjBPQkJZRUZIN1IKNXlDK1VlaElJUGV1TDhacXczUHpiZ2NaTUI4R0ExVWRJd1FZTUJhQUZIN1I0
eUMrVWVoSUlQZXVMOFpxdzNQegpjZ2NaTUE4R0ExVWRFd0VCL3dRRk1BTUJBZjh3RGdZRFZS
MFBBUUgvQkFRREFnR0dNQTBHQ1NxR1NJYjNEUUVCCkR3VUFBNElCQVFCRE52RDJWbTlzQT
VBOUFsT0pSOCtlbjVYejloWGN4SkI1cGh4Y1pROGpGb0cwNFZzaHZkMGUKTUVuVXJNY2ZGZ0laN
G5qTUtUUUNNNFpGVVBBaWV5THg0ZjUySHVEb3BwM2U1SnlJTWZXK0tGY05JcEt3Q3NhawpwU2
9LdElVT3NVSks3cUJWWnhjckl5ZVFWMnFjWU9lWmh0UzV3QnFJd09BaEZ3bENFVDdaZTU4UUhtUz
Q4c2xqCjVlVGtSaml2QWxFeHJGektjbGpDNGF4S1Fsbk92VkF6eitHbTMyVTB4UEJGNEJ5ZVBWeEN
KVUh3MVRzeVRtZWwKU3hORXA3eUhvWGN3bitmWG5hK3Q1SldoMWd4VVp0eTMKLS0tLS1FTkQ
gQ0VSVElGSUNBVEUtLS0tLQo=
    mode: 0644
    overwrite: true
    path: /etc/pki/ca-trust/source/anchors/examplecorp-ca.crt
```

머신의 신뢰 저장소는 노드의 신뢰 저장소 업데이트도 지원해야 합니다.

4.2.8. 갱신

RHCOS 노드에서 인증서를 자동 갱신할 수 있는 Operator가 없습니다.

참고

Red Hat에서는 CA 만료 시점을 모니터링하지 않습니다. 그러나 CA의 수명이 길기 때문에 일반적으로 문제가 되지 않습니다. 그러나 신뢰 번들을 정기적으로 업데이트해야 할 수도 있습니다.

4.3. 서비스 CA 인증서

4.3.1. 목적

service-ca는 OpenShift Container Platform 클러스터가 배포될 때 자체 서명된 CA를 만드는 Operator입니다.

4.3.2. 만료

사용자 정의 기간은 지원되지 않습니다. 자체 서명된 CA는 **tls.crt**(인증서), **tls.key**(개인 키) 및 **ca-bundle.crt**(CA 번들)의 규정된 이름 **service-ca/signing-key**로 보안에 저장됩니다.

다른 서비스에서는 **service.beta.openshift.io/serving-cert-secret-name: <secret name>**으로 서비스 리소스에 주석을 달아 서비스 제공 인증서를 요청할 수 있습니다. 이에 응답하여 Operator는 이름 지정된 보안에 대한 **tls.crt**로 새 인증서를, **tls.key**로 개인 키를 생성합니다. 이 인증서는 2년간 유효합니다.

다른 서비스에서는 서비스 CA에서 생성된 인증서의 검증을 지원하기 위해

service.beta.openshift.io/inject-cabundle: true로 주석을 달아 서비스 CA의 CA 번들을 API 서비스 또는 구성 맵 리소스에 삽입하도록 요청할 수 있습니다. 이에 응답하여 Operator는 현재 CA 번들을 API 서비스의 **CABundle** 필드에 쓰거나 **service-ca.crt**로 구성 맵에 씁니다.

OpenShift Container Platform 4.3.5부터 자동 순환이 지원되며 일부 4.2.z 및 4.3.z 릴리스로 백포트됩니다. 자동 순환을 지원하는 모든 릴리스에서 서비스 CA는 26개월 동안 유효하며 유효 기간이 13개월 미만으로 남아 있으면 자동으로 새로 고칩니다. 필요하면 서비스 CA를 직접 새로 고칠 수 있습니다.

26개월의 서비스 CA 만료 기간은 지원되는 OpenShift Container Platform 클러스터에 대한 업그레이드 예상 간격보다 길기 때문에 서비스 CA 인증서의 비컨트롤 플레인 소비자의 CA는 CA 순환 후와 순환 전 CA 만료 전에 새로 고칩니다.

주의

수동으로 순환된 서비스 CA는 이전 서비스 CA와의 신뢰를 유지 관리하지 않습니다. 클러스터의 Pod가 다시 시작되어 Pod가 새 서비스 CA에서 발급한 서비스 제공 인증서를 사용하고 있는지 확인할 때까지 일시적으로 서비스 중단이 발생할 수 있습니다.

4.3.3. 관리

이러한 인증서는 사용자가 아닌 시스템에서 관리합니다.

4.3.4. 서비스

서비스 CA 인증서를 사용하는 서비스는 다음과 같습니다.

- cluster-autoscaler-operator
- cluster-monitoring-operator
- cluster-authentication-operator
- cluster-image-registry-operator
- cluster-ingress-operator
- cluster-kube-apiserver-operator
- cluster-kube-controller-manager-operator
- cluster-kube-scheduler-operator
- cluster-networking-operator
- cluster-openshift-apiserver-operator
- cluster-openshift-controller-manager-operator
- cluster-samples-operator
- machine-config-operator

- console-operator
- insights-operator
- machine-api-operator
- operator-lifecycle-manager

이것은 포괄적인 목록이 아닙니다.

추가 리소스

- 서비스 제공 인증서를 직접 순환
- 서비스 제공 인증서 보안을 사용하여 서비스 트래픽 보안

4.4. 노드 인증서

4.4.1. 목적

노드 인증서는 클러스터를 통해 서명합니다. 노드 인증서는 부트스트랩 프로세스를 통해 생성된 인증 기관(CA)에서 제공합니다. 클러스터가 설치되면 노드 인증서가 자동으로 순환됩니다.

4.4.2. 관리

이러한 인증서는 사용자가 아닌 시스템에서 관리합니다.

추가 리소스

- 노드 작업

4.5. 부트스트랩 인증서

4.5.1. 목적

OpenShift Container Platform 4 이상에서 kubelet은 **/etc/kubernetes/kubeconfig**에 있는 부트스트랩 인증서를 사용하여 처음에 부트스트랩합니다. 그런 다음 부트스트랩 초기화 프로세스와 CSR을 생성하기 위한 kubelet 권한 부여가 이어집니다.

이 과정에서 kubelet은 부트스트랩 채널을 통해 통신하면서 CSR을 생성합니다. 컨트롤러 관리자는 CSR에 서명하여 kubelet을 통해 관리하는 인증서를 생성합니다.

4.5.2. 관리

이러한 인증서는 사용자가 아닌 시스템에서 관리합니다.

4.5.3. 만료

이 부트스트랩 CA는 10년 동안 유효합니다.

kubelet 관리형 인증서는 1년 동안 유효하며 1년 중 80% 정도되는 시점에 자동으로 순환됩니다.

4.5.4. 사용자 정의

부트스트랩 인증서를 사용자 정의할 수 없습니다.

4.6. ETCD 인증서

4.6.1. 목적

etcd 인증서는 etcd-signer를 통해 서명합니다. 노드 인증서는 부트스트랩 프로세스를 통해 생성된 인증 기관(CA)에서 제공합니다.

4.6.2. 만료

CA 인증서는 10년 동안 유효합니다. 피어, 클라이언트 및 서버 인증서는 3년간 유효합니다.

4.6.3. 관리

이러한 인증서는 사용자가 아닌 시스템에서 관리합니다.

4.6.4. 서비스

etcd 인증서는 etcd 멤버 피어 사이의 암호화된 통신 및 암호화된 클라이언트 트래픽에 사용됩니다. 다음 인증서는 etcd 및 etcd와 통신하는 다른 프로세스를 통해 생성되고 사용됩니다.

- 피어 인증서: etcd 멤버 간 통신에 사용됩니다.
- 클라이언트 인증서: 암호화된 서버-클라이언트 통신에 사용됩니다. 클라이언트 인증서는 현재 API 서버에서만 사용되며 프록시를 제외한 다른 서비스는 etcd에 직접 연결하지 않아야 합니다. 클라이언트 보안(**etcd-client**, **etcd-metric-client**, **etcd-metric-signer**, and **etcd-signer**)이 **openshift-config**, **openshift-monitoring** 및 **openshift-kube-apiserver** 네임스페이스에 추가됩니다.
- 서버 인증서: etcd 서버에서 클라이언트 요청을 인증하는 데 사용합니다.
- 지표 인증서: 모든 지표 소비자는 metric-client 인증서를 사용하여 프록시에 연결합니다.

추가 리소스

- 이전 클러스터 상태로 복원

4.7. OLM 인증서

4.7.1. 관리

OLM(OpenShift Lifecycle Manager) 구성요소(**olm-operator**, **catalog-operator**, **packageserver**, 및 **marketplace-operator**)의 모든 인증서는 시스템에서 관리합니다.

CSV(**ClusterServiceVersion**) 개체에 Webhook 또는 API 서비스가 포함된 Operator를 설치하면 OLM이 이러한 리소스의 인증서를 생성하고 회전합니다. **openshift-operator-lifecycle-manager** 네임스페이스의 리소스의 인증서는 OLM에서 관리합니다.

OLM은 프록시 환경에서 관리하는 Operator의 인증서를 업데이트하지 않습니다. 이러한 인증서는 서브스크립션 구성을 통해 사용자가 관리해야 합니다.

4.8. 기본 수신을 위한 사용자 제공 인증서

4.8.1. 목적

애플리케이션은 일반적으로 **<route_name>.apps.<cluster_name>.<base_domain>**에서 노출됩니다. **<cluster_name>** 및 **<base_domain>** 은 설치 구성 파일에서 가져옵니다. **<route_name>** 은 지정된 경우 경로의 호스트 필드 또는 경로 이름입니다. 예를 들어 **hello-openshift-default.apps.username.devcluster.openshift.com.hello-openshift** 는 경로의 이름이고 경로는 기본 네임스페이스에 있습니다. 클러스터 관리 CA 인증서를 클라이언트에 배포할 필요없이 클라이언트가 애플리케이션에 액세스하게 합니다. 애플리케이션 콘텐츠를 제공할 때 관리자는 사용자 정의 기본 인증서를 설정해야 합니다.

주의

사용자가 사용자 정의 기본 인증서를 구성할 때까지 Ingress Operator가 자리 표시자로 사용될 Ingress Controller의 기본 인증서를 생성합니다. 프로덕션 클러스터에서는 operator가 생성한 기본 인증서를 사용하지 마십시오.

4.8.2. 위치

사용자 제공 인증서는 **openshift-ingress** 네임스페이스에 **tls** 유형 **Secret** 리소스로 제공되어야 합니다. 사용자 제공 인증서를 사용할 수 있도록 **openshift-ingress-operator** 네임스페이스에서 **IngressController** CR을 업데이트합니다. 이 프로세스에 대한 자세한 내용은 사용자 정의 기본 인증서 설정을 참조하십시오.

4.8.3. 관리

사용자 제공 인증서는 사용자가 관리합니다.

4.8.4. 만료

사용자 제공 인증서는 사용자가 관리합니다.

4.8.5. 서비스

클러스터에 배포된 애플리케이션에서는 기본 수신에 사용자 제공 인증서를 사용합니다.

4.8.6. 사용자 정의

필요에 따라 사용자 관리 인증서가 포함된 보안을 업데이트합니다.

추가 리소스

- 기본 수신 인증서 교체

4.9. 수신 인증서

4.9.1. 목적

Ingress Operator에서는 다음을 위해 인증서를 사용합니다.

- Prometheus의 지표에 대한 액세스 보안
- 경로에 대한 액세스 보안

4.9.2. 위치

Ingress Operator 및 Ingress Controller 지표에 대한 액세스를 보호하기 위해 Ingress Operator에서는 서비스 제공 인증서를 사용합니다. Operator가 **service-ca** 컨트롤러에서 고유 지표에 맞는 인증서를 요청하고, **service-ca** 컨트롤러가 **metrics-tls**라는 보안의 인증서를 **openshift-ingress-operator** 네임스페이스에 둡니다. 또한 Ingress Operator에서 각 Ingress Controller의 인증서를 요청하고 **service-ca** 컨트롤러에서 **router-metrics-certs-<name>**라는 보안에 인증서를 넣습니다. 여기서 **<name>**은 **openshift-ingress** 네임스페이스의 Ingress Controller의 이름입니다.

각 Ingress Controller에는 고유 인증서를 지정하지 않는 보안 경로에 사용하는 기본 인증서가 있습니다. 사용자 정의 인증서를 지정하지 않으면 Operator에서 기본적으로 자체 서명된 인증서를 사용합니다. Operator는 자체 서명된 서명 인증서를 사용하여 생성하는 기본 인증서에 서명합니다. Operator는 이 서명 인증서를 생성하여 **openshift-ingress-operator** 네임스페이스의 **router-ca**라는 보안에 둡니다. Operator가 기본 인증서를 생성하고 **openshift-ingress** 네임스페이스의 **router-certs-<name>**라는 보안에 기본 인증서를 둡니다(여기서 **<name>**은 Ingress Controller의 이름임).

주의

사용자가 사용자 정의 기본 인증서를 구성할 때까지 Ingress Operator가 자리 표시자로 사용될 Ingress Controller의 기본 인증서를 생성합니다. 프로덕션 클러스터에서는 operator가 생성한 기본 인증서를 사용하지 마십시오.

4.9.3. 워크플로

그림 4.1. 사용자 정의 인증서 워크플로

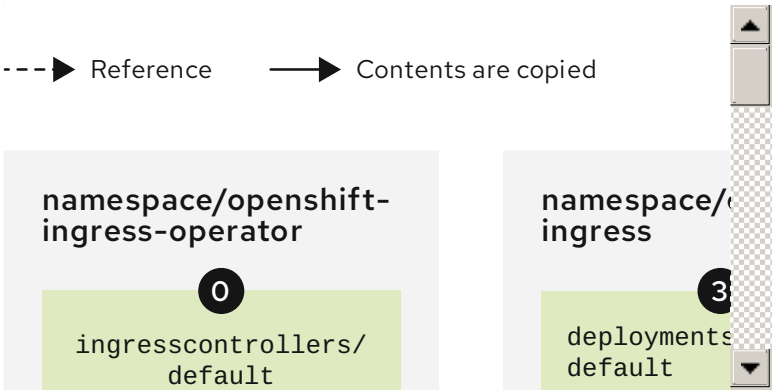
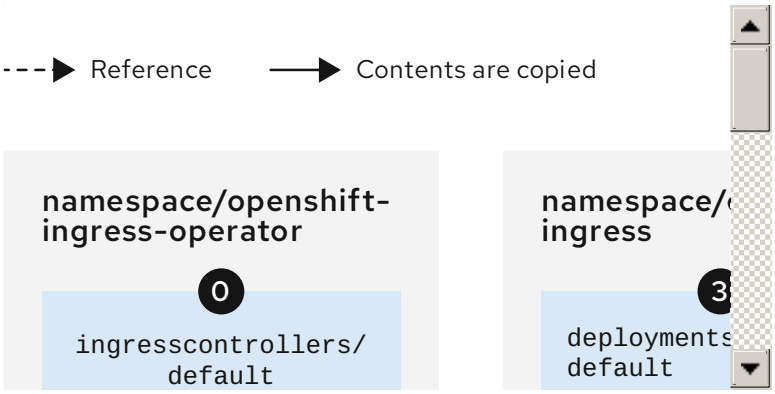


그림 4.2. 기본 인증서 워크플로



0 **defaultCertificate** 필드가 비어 있으면 Ingress Operator에서 자체 서명된 CA를 사용하여 지정된 도메인의 제공 인증서를 생성합니다.

1 Ingress Operator에서 생성한 기본 CA 인증서 및 키. Operator 생성 기본 제공 인증서에 서명하는 데 사용합니다.

2 기본 워크플로에서는 Ingress Operator가 작성하고 생성된 기본 CA 인증서를 사용하여 서명한 와일드카드 기본 제공 인증서입니다. 사용자 정의 워크플로에서 이 인증서는 사용자 제공 인증서입니다.

3 라우터 배포. **secrets/router-certs-default**의 인증서를 기본 프론트 엔드 서버 인증서로 사용합니다.

4 기본 워크플로에서 와일드카드 기본 제공 인증서(공용 및 개인용 부분)의 콘텐츠가 여기에 복사되므로 OAuth 통합이 가능합니다. 사용자 정의 워크플로에서 이 인증서는 사용자 제공 인증서입니다.

5 기본 제공 인증서의 공용(인증서) 부분입니다. **configmaps/router-ca** 리소스를 교체합니다.

6 사용자는 **ingresscontroller** 제공 인증서에 서명한 CA 인증서로 클러스터 프록시 구성을 업데이트합니다. 그러면 **auth**, **console** 및 레지스트리와 같은 구성요소가 제공 인증서를 신뢰할 수 있습니다.

7 사용자 번들이 제공되지 않은 경우 RHCOS(Red Hat Enterprise Linux CoreOS) 및 사용자 제공 CA 번들 또는 RHCOS 전용 번들을 포함하는 클러스터 전체의 신뢰할 수 있는 CA 번들입니다.

8 사용자 정의 인증서로 구성된 **ingresscontroller**를 신뢰하도록 다르 구성요소(예: **auth** 및 **console**)에 지시하는 사용자 정의 CA 인증서 번들입니다.

9 **trustedCA** 필드는 사용자 제공 CA 번들을 참조하는 데 사용됩니다.

10 Cluster Network Operator에서 신뢰할 수 있는 CA 번들을 **proxy-ca** 구성 맵에 삽입합니다.

11 OpenShift Container Platform 4.6 이상에서는 **default-ingress-cert**를 사용합니다.

4.9.4. 만료

Ingress Operator 인증서의 만료 기간은 다음과 같습니다.

- **service-ca** 컨트롤러가 생성하는 메트릭 인증서의 만료 날짜는 생성일로부터 2년입니다.

- Operator 서명 인증서의 만료 날짜는 생성일로부터 2년입니다.
- Operator가 생성하는 기본 인증서의 만료 날짜는 생성일로부터 2년입니다.

Ingress Operator 또는 **service-ca** 컨트롤러에서 생성하는 인증서에 사용자 정의 만료 기간을 지정할 수 없습니다.

Ingress Operator 또는 **Service-CA** 컨트롤러가 생성하는 인증서에 맞게 OpenShift Container Platform을 설치하면 만료 조건을 지정할 수 없습니다.

4.9.5. 서비스

Prometheus에서는 메트릭을 보호하는 인증서를 사용합니다.

Ingress Operator에서는 서명 인증서를 사용하여 사용자 정의 기본 인증서를 설정하지 않은 Ingress Controller용으로 생성한 기본 인증서에 서명합니다.

보안 경로를 사용하는 클러스터 구성요소는 기본 Ingress Controller의 기본 인증서를 사용할 수 있습니다.

보안 경로를 통해 클러스터로 수신할 때는 경로에서 고유 인증서를 지정하지 않는 한 경로에 액세스하는 Ingress Controller의 기본 인증서를 사용합니다.

4.9.6. 관리

수신 인증서는 사용자가 관리합니다. 자세한 내용은 기본 수신 인증서 교체 를 참조하십시오.

4.9.7. 갱신

service-ca 컨트롤러에서는 발급한 인증서가 자동으로 순환됩니다. 그러나 **oc delete secret <secret>**을 사용하여 서비스 제공 인증서를 직접 순환할 수 있습니다.

Ingress Operator에서는 자체 서명 인증서 또는 생성된 기본 인증서가 순환되지 않습니다. Operator가 생성한 기본 인증서는 구성하는 사용자 정의 기본 인증서의 자리 표시자로 사용됩니다.

4.10. 모니터링 및 클러스터 로깅 OPERATOR 구성요소 인증서

4.10.1. 만료

모니터링 구성요소는 서비스 CA 인증서로 트래픽을 보호합니다. 이 인증서는 2년 동안 유효하며 서비스 CA 순환 시 13개월마다 자동으로 교체됩니다.

인증서가 **openshift-monitoring** 또는 **openshift-logging** 네임스페이스에 있으면 시스템 관리 및 순환이 자동으로 이루어집니다.

4.10.2. 관리

이러한 인증서는 사용자가 아닌 시스템에서 관리합니다.

4.11. 컨트롤 플레인 인증서

4.11.1. 위치

컨트롤 플레인 인증서는 다음 네임스페이스에 포함되어 있습니다.

- openshift-config-managed
- openshift-kube-apiserver
- openshift-kube-apiserver-operator
- openshift-kube-controller-manager
- openshift-kube-controller-manager-operator
- openshift-kube-scheduler

4.11.2. 관리

컨트롤 플레인 인증서는 시스템에서 관리하고 자동으로 순환됩니다.

드문 경우지만 컨트롤 플레인 인증서가 만료된 경우 만료된 컨트롤 플레인 인증서에서 복구 를 참조하십시오.

5장. COMPLIANCE OPERATOR

5.1. COMPLIANCE OPERATOR 릴리스 정보

OpenShift Container Platform 관리자는 Compliance Operator를 통해 클러스터의 필수 규정 준수 상태를 설명하고 격차에 대한 개요와 문제를 해결하는 방법을 제공할 수 있습니다.

이 릴리스 노트는 OpenShift Container Platform의 Compliance Operator 개발을 추적합니다.

Compliance Operator에 대한 개요는 Compliance Operator 이해를 참조하십시오.

5.1.1. OpenShift Compliance Operator 0.1.53

OpenShift Compliance Operator 0.1.53에 다음 권고를 사용할 수 있습니다.

- RHBA-2022:5537 - OpenShift Compliance Operator 버그 수정 업데이트

5.1.1.1. 버그 수정

- 이전 버전에서는 **ocp4-kubelet-enable-streaming-connections** 규칙에 잘못된 변수 비교가 포함되어 잘못된 검사 결과가 발생했습니다. 이제 **streamingConnectionIdleTimeout** 을 설정할 때 Compliance Operator에서 정확한 검사 결과를 제공합니다. (BZ#2069891)
- 이전에는 IBM Z 아키텍처에서 **/etc/openvswitch/conf.db** 의 그룹 소유권이 올바르지 않아 **ocp4-cis-worker-worker-file-groupowner-ovs-conf-db** 검사 실패가 발생했습니다. 이제 IBM Z 아키텍처 시스템에서 검사에 **NOT-APPLICABLE** 이 표시됩니다. (BZ#2072597)
- 이전에는 배포의 보안 컨텍스트 제약 조건(SCC) 규칙에 대한 불완전한 데이터로 인해 **FAIL** 상태에서 보고되는 **ocp4-cis-scc-limit-container-capabilities** 규칙이 보고되었습니다. 이제 최종 결과는 사람의 개입이 필요한 다른 검사와 일치합니다. **(BZ#2077916)**
- **이전 버전에서는 다음 규칙이 API 서버 및 TLS 인증서 및 키의 추가 구성 경로를 고려하지 않아 인증서 및 키가 올바르게 설정된 경우에도 오류가 보고되었습니다.**
  - **ocp4-cis-api-server-kubelet-client-cert**
  - **ocp4-cis-api-server-kubelet-client-key**
  - **ocp4-cis-kubelet-configure-tls-cert**
  - **ocp4-cis-kubelet-configure-tls-key**

이제 규칙 보고서가 kubelet 구성 파일에 지정된 레거시 파일 경로를 정확하게 보고하고 관

찰합니다. (BZ#2079813)

- 이전 버전에서는 content_rule_oauth_oauthclient_inactivity_timeout 규칙에 따라 시간 초과에 대한 규정 준수를 평가할 때 배포에 설정된 구성 가능한 시간 초과가 포함되지 않았습니다. 이로 인해 시간 초과가 유효한 경우에도 규칙이 실패합니다. 이제 Compliance Operator에서 var_oauth_inactivity_timeout 변수를 사용하여 유효한 시간 초과 길이를 설정합니다. (BZ#2081952)

- 이전에는 Compliance Operator에서 권한 있는 사용에 대해 적절히 레이블이 지정되지 않은 네임스페이스에 관리 권한을 사용했기 때문에 Pod 보안 수준 위반과 관련된 경고 메시지가 표시되었습니다. 이제 Compliance Operator에 권한을 위반하지 않고 결과에 액세스할 수 있도록 적절한 네임스페이스 레이블 및 권한 조정이 있습니다. (BZ#2088202)

- 이전 버전에서는 rhcos4-high-master-sysctl-kernel-yama-ptrace-scope 및 rhcos4-sysctl-kernel-core-pattern 에 대한 자동 수정을 적용하면 수정되었지만 해당 규칙이 나중에 검사 결과에 실패했습니다. 이제 규칙에서 수정 사항이 적용된 후에도 PASS 를 정확하게 보고합니다. (BZ#2094382)

- 이전에는 메모리 부족 예외로 인해 Compliance Operator가 CrashLoopBackoff 상태로 실패했습니다. 이제 Compliance Operator가 메모리의 대규모 머신 구성 데이터 세트를 올바르게 처리하도록 개선되었습니다. (BZ#2094854)

5.1.1.2. 알려진 문제

- ScanSettingBinding 오브젝트 내에 "debug":true 가 설정된 경우 ScanSettingBinding 오브젝트에서 생성한 Pod는 해당 바인딩이 삭제될 때 제거되지 않습니다. 이 문제를 해결하려면 다음 명령을 실행하여 나머지 Pod를 삭제합니다.

```
$ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
```

(BZ#2092913)

5.1.2. OpenShift Compliance Operator 0.1.52

OpenShift Compliance Operator 0.1.52에서 다음 권고를 사용할 수 있습니다.

- RHBA-2022:4657 - OpenShift Compliance Operator 버그 수정 업데이트

5.1.2.1. 새로운 기능 및 개선 사항

- 이제 OpenShift Container Platform 환경에서 FedRAMP 높은 SCAP 프로필을 사용할 수 있습니다. 자세한 내용은 지원되는 규정 준수 프로필을 참조하십시오.

5.1.2.2. 버그 수정

- 이전에는 DAC_OVERRIDE 기능이 삭제된 보안 환경의 마운트 권한 문제로 인해 OpenScap 컨테이너가 충돌했습니다. 이제 실행 가능한 마운트 권한이 모든 사용자에게 적용됩니다. (BZ#2082151)

- 이전에는 규정 준수 규칙 ocp4-configure-network-policies 를 MANUAL 으로 구성할 수 있었습니다. 이제 규정 준수 규칙 ocp4-configure-network-policies 가 AUTOMATIC 으로 설정됩니다. (BZ#2072431)

- 이전에는 Compliance Operator 검사 Pod가 검사 후 제거되지 않았기 때문에 Cluster Autoscaler가 축소되지 않았습니다. 이제 디버깅 목적으로 명시적으로 저장되지 않는 한 Pod는 기본적으로 각 노드에서 제거됩니다. (BZ#2075029)

- 이전에는 KubeletConfig 에 Compliance Operator를 적용하면 Machine Config Pools가 너무 일찍 일시 정지되지 않아 노드가 NotReady 상태가되었습니다. 이제 Machine Config Pools가 적절하게 일시 중지되지 않고 노드가 올바르게 작동합니다. (BZ#2071854)

- 이전에는 Machine Config Operator에서 최신 릴리스에서 url로 인코딩된 코드 대신 base64 를 사용하여 Compliance Operator 수정이 실패했습니다. 이제 Compliance Operator 에서 base64 및 url-encoded Machine Config 코드를 모두 처리하도록 인코딩을 확인하고 수정이 올바르게 적용됩니다. (BZ#2082431)

5.1.2.3. 알려진 문제

- ScanSettingBinding 오브젝트 내에 "debug":true 가 설정된 경우 ScanSettingBinding 오브젝트에서 생성한 Pod는 해당 바인딩이 삭제될 때 제거되지 않습니다. 이 문제를 해결하려면 다음 명령을 실행하여 나머지 Pod를 삭제합니다.

```
$ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
```

(BZ#2092913)

5.1.3. OpenShift Compliance Operator 0.1.49

OpenShift Compliance Operator 0.1.49에 다음 권고를 사용할 수 있습니다.

- RHBA-2022:1148 - OpenShift Compliance Operator bug fix and enhancement update

5.1.3.1. 버그 수정

- 이전에는 **openshift-compliance** 콘텐츠에 네트워크 유형에 대한 플랫폼별 검사가 포함되지 않았습니다. 그 결과 네트워크 구성에 따라 적용되지 않고 **OVN-** 및 **SDN** 관련 검사에 실패한 것으로 표시됩니다. 이제 새로운 규칙에 네트워킹 규칙에 대한 플랫폼 검사를 포함하므로 네트워크별 검사를 보다 정확하게 평가할 수 있습니다. (BZ#1994609)

- 이전에는 **ocp4-moderate-routes-protected-by-tls** 규칙에서 연결 보안 **SSL TLS** 프로토콜에도 불구하고 검사에 실패한 **TLS** 설정을 잘못 확인했습니다. 이제 검사에서 네트워킹 지침 및 프로필 권장 사항과 일치하는 **TLS** 설정을 올바르게 평가합니다. (BZ#2002695)

- 이전에는 **ocp-cis-configure-network-policies-namespace** 가 네임스페이스를 요청할 때 **pagination**을 사용했습니다. 이로 인해 배포에 **500**개 이상의 네임스페이스 목록이 잘렸기 때문에 규칙이 실패했습니다. 이제 전체 네임스페이스 목록이 요청되고 구성된 네트워크 정책을 확인하는 규칙이 **500**개 이상의 네임스페이스를 사용하는 배포에 적용됩니다. (BZ#2038909)

- 이전에는 **sshd jinja** 매크로를 사용한 수정 사항이 특정 **sshd** 구성으로 하드 코딩되었습니다. 결과적으로 규칙이 검사한 내용과 구성이 일치하지 않았으며 검사가 실패합니다. 이제 **sshd** 구성이 매개 변수화되어 규칙이 성공적으로 적용됩니다. (BZ#2049141)

- 이전에는 **ocp4-cluster-version-operator-verify-integrity** 가 항상 **Cluter Version Operator (CVO)** 기록의 첫 번째 항목을 확인했습니다. 결과적으로 후속 버전의 **{product-name}** 이 확인되는 경우 업그레이드가 실패합니다. 이제 **ocp4-cluster-version-operator-verify-integrity** 에 대한 규정 준수 확인 결과가 확인된 버전을 감지할 수 있으며 **CVO** 기록으로 정확합니다. (BZ#2053602)

- 이전에는 **ocp4-api-server-no-adm-ctrl-plugins-disabled** 규칙에서 빈 승인 컨트롤러 플러그인 목록을 확인하지 않았습니다. 결과적으로 모든 승인 플러그인이 활성화된 경우에도 규칙이 항상 실패합니다. 이제 **ocp4-api-server-no-adm-ctrl-plugins-disabled** 규칙을 더 강력하게 확인하면 모든 승인 컨트롤러 플러그인이 활성화된 상태에서 정확하게 전달됩니다. (BZ#2058631)

- 이전에는 검사에서 **Linux** 작업자 노드에 대해 실행할 플랫폼 검사를 포함하지 않았습니다. 결과적으로 **Linux** 기반이 아닌 작업자 노드에 대해 검사를 실행하면 검사 루프가 끝나지 않았습

니다. 이제 검사가 플랫폼 유형 및 라벨에 따라 적절하게 스케줄링되고 완전히 성공적으로 예약됩니다. (BZ#2056911)

5.1.4. OpenShift Compliance Operator 0.1.48

다음 권고는 OpenShift Compliance Operator 0.1.48에서 사용할 수 있습니다.

- RHBA-2022:0416 - OpenShift Compliance Operator bug fix and enhancement update

5.1.4.1. 버그 수정

- 이전 버전에서는 OCI(Extended Open Vulnerability and assess Language) 정의와 관련된 일부 규칙에는 checkType 의 None 이 있었습니다. 이는 Compliance Operator가 규칙을 구문 분석할 때 확장된 OVAL 정의를 처리하지 않았기 때문입니다. 이번 업데이트를 통해 확장된 OVAL 정의의 콘텐츠가 구문 분석되므로 이러한 규칙에는 Node 또는 Platform 중 하나의 checkType 이 있습니다. (BZ#2040282)

- 이전 버전에서는 KubeletConfig 에 대해 수동으로 생성된 MachineConfig 오브젝트를 통해 KubeletConfig 오브젝트가 수정을 위해 생성되지 않아 수정이 Pending 상태로 남아 있었습니다. 이번 릴리스에서는 KubeletConfig 용으로 수동으로 생성된 MachineConfig 오브젝트가 있는지 여부에 관계없이 문제 해결을 통해 KubeletConfig 오브젝트가 생성됩니다. 결과적으로 KubeletConfig 수정이 예상대로 작동합니다. (BZ#2040401)

5.1.5. OpenShift Compliance Operator 0.1.47

OpenShift Compliance Operator 0.1.47에 다음 권고를 사용할 수 있습니다.

- RHBA-2022:0014 - OpenShift Compliance Operator 버그 수정 및 개선 사항 업데이트

5.1.5.1. 새로운 기능 및 개선 사항

- Compliance Operator는 PCI DSS(Payment Card Industry Data Security Standard)에 대한 다음 규정 준수 벤치마크를 지원합니다.
  - ocp4-pci-dss

- ocp4-pci-dss-node

- FedRAMP 보통 영향 수준에 대한 추가 규칙 및 수정은 OCP4-moderate, OCP4-moderate-node 및 rhcos4-moderate 프로필에 추가됩니다.

- 노드 수준 프로필에서 KubeletConfig 수정을 사용할 수 있습니다.

5.1.5.2. 버그 수정

- 이전 버전에서는 클러스터가 OpenShift Container Platform 4.6 또는 이전 버전을 실행하는 경우 적절한 프로필에 대해 USBGuard 관련 규칙에 대한 수정이 실패했습니다. 이는 Compliance Operator에서 생성한 수정이 드롭인 디렉터리를 지원하지 않는 이전 버전의 USBGuard를 기반으로 하기 때문입니다. 이제 OpenShift Container Platform 4.6을 실행하는 클러스터에는 USBGuard 관련 규칙에 대한 잘못된 수정이 생성되지 않습니다. 클러스터가 OpenShift Container Platform 4.6을 사용하는 경우 USBGuard 관련 규칙에 대한 수정을 수동으로 생성해야 합니다.

  또한 최소 버전 요구 사항을 충족하는 규칙 전용 수정이 생성됩니다. (BZ#1965511)

- 이전 버전에서는 수정을 렌더링할 때 규정 준수 연산자는 너무 엄격한 정규 표현식을 사용하여 해결 방법의 형식이 올바른지 확인했습니다. 그 결과 sshd_config 를 렌더링하는 것과 같은 일부 수정이 정규 표현식 검사를 통과하지 않으므로 생성되지 않았습니다. 정규 표현식은 불필요하고 제거된 것으로 확인되었습니다. 이제 수정 사항이 올바르게 렌더링됩니다. (BZ#2033009)

5.1.6. OpenShift Compliance Operator 0.1.44

OpenShift Compliance Operator 0.1.44에 대해 다음 권고를 사용할 수 있습니다.

- RHBA-2021:4530 - OpenShift Compliance Operator 버그 수정 및 개선 사항 업데이트

5.1.6.1. 새로운 기능 및 개선 사항

- 이번 릴리스에서는 ComplianceScan, ComplianceSuite 및 ScanSetting CR에 strictNodeScan 옵션이 추가되었습니다. 이 옵션은 노드에서 검색을 예약할 수 없는 경우 오류가 발생한 이전 동작과 일치하는 true로 기본 설정됩니다. 옵션을 false로 설정하면 Compliance Operator가 스케줄링 검사에 대해 더 관대해질 수 있습니다. 임시 노드가 있는 환경에서는 strictNodeScan 값을 false로 설정할 수 있으므로 클러스터의 일부 노드를 예약할 수 없는 경우에도 규정 준수 검사를 진행할 수 있습니다.

- 이제 **ScanSetting** 오브젝트의 **nodeSelector** 및 **tolerations** 속성을 구성하여 결과 서버 워크로드를 예약하는 데 사용되는 노드를 사용자 지정할 수 있습니다. 이러한 속성은 **PV** 스토리지 볼륨을 마운트하고 원시 자산 보고 형식**(ARF)** 결과를 저장하는 데 사용되는 **Pod**인 **ResultServer Pod**를 배치하는 데 사용됩니다. 이전에는 **nodeSelector** 및 **tolerations** 매개변수가 컨트롤 플레인 노드 중 하나를 선택하고 **node-role.kubernetes.io/master taint**를 허용하는 것으로 기본 설정되었습니다. 이는 컨트롤 플레인 노드가 **PV**를 마운트할 수 없는 환경에서는 작동하지 않았습니다. 이 기능을 사용하면 해당 환경에서 노드를 선택하고 다른 테인트를 허용할 수 있습니다.

- **Compliance Operator**에서 **KubeletConfig** 오브젝트를 수정할 수 있습니다.

- 이제 오류 메시지가 포함된 주석이 추가되어 콘텐츠 개발자가 클러스터에 없는 오브젝트와 가져올 수 없는 개체를 구분할 수 있습니다.

- 이제 **rule** 오브젝트에 **checkType** 및 **description** 이라는 두 개의 새 속성이 포함됩니다. 이러한 속성을 사용하면 규칙과 노드 점검 또는 플랫폼 점검이 적용되는지 확인하고 규칙의 기능을 검토할 수도 있습니다.

- 이번 개선된 기능을 통해 맞춤형 프로필을 생성하기 위해 기존 프로필을 확장해야 하는 요구사항이 제거됩니다. 즉 **TailoredProfile CRD**의 **extends** 필드가 더 이상 필수가 아닙니다. 이제 규칙 오브젝트 목록을 선택하여 맞춤형 프로필을 생성할 수 있습니다. **compliance.openshift.io/product-type:** 주석을 설정하거나 **TailoredProfile CR**에 **-node** 접미사를 설정하여 프로필이 노드 또는 플랫폼에 적용되는지 여부를 선택해야 합니다.

- 이번 릴리스에서 **Compliance Operator**는 테인트와 관계없이 모든 노드에서 검사를 예약할 수 있습니다. 이전에는 검사 **Pod**에서 **node-role.kubernetes.io/master taint**만 허용했습니다. 즉, 테인트가 없는 노드에서나 **node-role.kubernetes.io/master** 테인트가 있는 노드에서만 실행되었습니다. 노드에 사용자 정의 테인트를 사용하는 배포에서는 해당 노드에 검사가 예약되지 않았습니다. 이제 검사 **Pod**에서 모든 노드 테인트를 허용합니다.

- 이 릴리스에서 **Compliance Operator**는 다음과 같은**NERC(North American Power Reliability Corporation)** 보안 프로필을 지원합니다.

  - **ocp4-nerc-cip**

  - **ocp4-nerc-cip-node**

- rhcos4-nerc-cip

- 이번 릴리스에서 **Compliance Operator**는 **Red Hat OpenShift -** 노드 수준**, ocp4-moderate-node,** 보안 프로파일에 대한 **NIST 800-53 Moderate-Impact Baseline**을 지원합니다.

5.1.6.2. 템플릿 및 변수 사용

- 이번 릴리스에서는 해결 템플릿에서 다중 값 변수를 허용합니다.

- 이번 업데이트를 통해 **Compliance Operator**는 규정 준수 프로필에 설정된 변수를 기반으로 수정을 변경할 수 있습니다. 이는 시간 제한**, NTP** 서버 호스트 이름 또는 유사한 배포별 값을 포함하는 수정에 유용합니다. 또한 **ComplianceCheckResult** 오브젝트에서 검사에 사용된 변수를 나열하는 **compliance.openshift.io/check-has-value** 레이블을 사용합니다.

5.1.6.3. 버그 수정

- 이전에는 검사를 수행하는 동안 **Pod**의 스캐너 컨테이너 중 하나에서 예기치 않은 종료가 발생했습니다. 이번 릴리스에서 **Compliance Operator**는 최신 **OpenSCAP** 버전 **1.3.5**를 사용하여 충돌을 방지합니다.

- 이전 버전에서는 **autoReplyRemediations**를 사용하여 수정을 적용하면 클러스터 노드 업데이트가 트리거되었습니다. 일부 수정에 필요한 입력 변수가 모두 포함되지 않은 경우 이로 인해 중단되었습니다. 이제 수정에 필요한 입력 변수가 하나 이상 누락된 경우 **NeedsReview** 상태가 할당됩니다. 하나 이상의 수정이 **needsReview** 상태에 있는 경우 머신 구성 풀은 일시 중지된 상태로 남아 있으며 모든 필수 변수가 설정될 때까지 수정이 적용되지 않습니다. 이렇게 하면 노드 중단을 최소화할 수 있습니다.

- **Prometheus** 지표에 사용되는 **RBAC** 역할 및 역할 바인딩이 **'ClusterRole'** 및 **'ClusterRoleBinding'**으로 변경되어 사용자 지정 없이 모니터링이 작동하는지 확인합니다.

- 이전에는 프로필을 구문 분석하는 중에 오류가 발생하면 규칙 또는 변수 오브젝트가 프로필에서 제거되고 삭제되었습니다. 이제 구문 분석 중에 오류가 발생하면 **profileparser**는 구문 분석이 완료될 때까지 오브젝트가 삭제되지 않도록 임시 주석으로 오브젝트에 주석을 추가합니다. (BZ#1988259)

- 이전에는 맞춤형 프로필에서 제목 또는 설명이 누락된 경우 오류가 발생했습니다. **XCCDF** 표준에는 맞춤형 프로필에 대한 제목과 설명이 필요하므로 이제 **TailoredProfile CR**에 제목 및 설명을 설정해야 합니다.

- 이전에는 맞춤형 프로필을 사용할 때 특정 선택 세트만 사용하여 **TailoredProfile** 변수 값을 설정할 수 있었습니다. 이제 이 제한이 제거되고 **TailoredProfile** 변수를 임의의 값으로 설정할 수 있습니다.

5.1.7. Compliance Operator 0.1.39의 릴리스 정보

OpenShift Compliance Operator 0.1.39에 대해 다음 권고를 사용할 수 있습니다.

- **RHBA-2021:3214 - OpenShift Compliance Operator** 버그 수정 및 개선 사항 업데이트

5.1.7.1. 새로운 기능 및 개선 사항

- 이전에는 **Compliance Operator**에서 **PCI DSS(Payment Card Industry Data Security Standard)** 참조를 구문 분석할 수 없었습니다. 이제 **Operator**에서 **PCI DSS** 프로필과 함께 제공되는 컴플라이언스 콘텐츠를 구문 분석할 수 있습니다.

- 이전에는 **Compliance Operator**에서 보통 프로필에서 **AU-5** 제어 규칙을 실행할 수 없었습니다. 이제 **Prometheusrules.monitoring.coreos.com** 오브젝트를 읽고 보통 프로필에서 **AU-5** 제어를 포함하는 규칙을 실행할 수 있도록 **Operator**에 권한이 추가됩니다.

5.1.8. 추가 리소스

- **Compliance Operator** 이해

5.2. 지원되는 규정 준수 프로필

Compliance Operator (CO) 설치의 일부로 몇 가지 프로필을 사용할 수 있습니다.

5.2.1. 규정 준수 프로필

Compliance Operator는 다음 규정 준수 프로필을 제공합니다.

표 **5.1.** 지원되는 규정 준수 프로필

Profile	프로파일 제목	Compliance Operator 버전	업계 규정 준수 벤치마크	지원되는 아키텍처
ocp4-cis	CIS Red Hat OpenShift Container Platform 4 벤치마크	0.1.39+	CIS 벤치마크 ™ footnote:cisbenchmark[CIS RedHat OpenShift Container Platform v4 Benchmarks를 찾으려면 CIS 벤치마크로 이동하여 검색 상자에 **Kubernetes** 를 입력합니다. **Kubernetes** 를 클릭한 다음 **최신 CIS 벤치마크 다운로드를** 클릭합니다. 여기에서 등록하면 벤치마크를 다운로드할 수 있습니다.]	**x86_64 ppc64le s390x**
ocp4-cis-node	CIS Red Hat OpenShift Container Platform 4 벤치마크	0.1.39+	CIS Benchmarks ™ footnote:cisbenchmark[]	**x86_64 ppc64le s390x**
ocp4-e8	austriental Essential Essential Eight (ACSC)	0.1.39+	ACSC Hardening Linux Workstations 및 Servers	**x86_64**
ocp4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Platform level	0.1.39+	NIST SP-800-53 릴리스 검색	**x86_64**
rhcos4-e8	austriental Essential Essential Eight (ACSC)	0.1.39+	ACSC Hardening Linux Workstations 및 Servers	**x86_64**
rhcos4-moderate	Red Hat Enterprise Linux CoreOS용 NIST 800-53 Moderate-Impact Baseline	0.1.39+	NIST SP-800-53 릴리스 검색	**x86_64**
ocp4-moderate-node	Red Hat OpenShift용 NIST 800-53 Moderate-Impact Baseline - 노드 수준	0.1.44+	NIST SP-800-53 릴리스 검색	**x86_64**
ocp4-nerc-cip	Red Hat OpenShift Container Platform - 플랫폼 수준에 대한 북아메리카의 CARC(Critical Infrastructure Protection)CIP(Critical Infrastructure Protection) 표준 프로파일	0.1.44+	NERC CIP 표준	**x86_64**

Profile	프로파일 제목	Compliance Operator 버전	업계 규정 준수 벤치마크	지원되는 아키텍처
ocp4-nerc-cip-node	Red Hat OpenShift Container Platform - 노드 수준에 대한 노아메리카 중개체국(NERC) Critical Infrastructure Protection (CIP)의 보안 보안 표준 프로파일	0.1.44+	NERC CIP 표준	**x86_64**
rhcos4-nerc-cip	북아메리카의 NERC(ReliabilityECDHE) CIP(Critical Infrastructure Protection)CIP(Critical Infrastructure Protection) Red Hat Enterprise Linux CoreOS의 CIP(Critical Reliability Protection) 표준 프로필	0.1.44+	NERC CIP 표준	**x86_64**
ocp4-pci-dss	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	0.1.47+	PCI 보안 표준 ®Center 문서 라이브러리	**x86_64**
ocp4-pci-dss-node	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	0.1.47+	PCI 보안 표준 ®Center 문서 라이브러리	**x86_64**
ocp4-high	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - 플랫폼 수준	0.1.52+	NIST SP-800-53 릴리스 검색	**x86_64**
ocp4-high-node	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - 노드 수준	0.1.52+	NIST SP-800-53 릴리스 검색	**x86_64**
rhcos4-high	NIST 800-53 High-Impact Baseline for Red Hat Enterprise Linux CoreOS	0.1.52+	NIST SP-800-53 릴리스 검색	**x86_64**

5.2.2. 추가 리소스

- 시스템에서 사용 가능한 규정 준수 프로필 보기에 대한 자세한 내용은 **Compliance Operator 이해의 Compliance Operator 프로필을** 참조하십시오.

5.3. COMPLIANCE OPERATOR 설치

Compliance Operator를 사용하려면 먼저 클러스터에 배포되었는지 확인해야 합니다.

5.3.1. 웹 콘솔을 통해 Compliance Operator 설치

사전 요구 사항

- **admin** 권한이 있어야 합니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에서 **Operator → OperatorHub**로 이동합니다.
2. **Compliance Operator**를 검색한 다음 설치를 클릭합니다.
3. 기본 설치 모드 및 네임스페이스를 계속 선택하여 **Operator**가 **openshift-compliance** 네임스페이스에 설치되도록 합니다.
4. 설치를 클릭합니다.

검증

설치에 성공했는지 확인하려면 다음을 수행하십시오.

1. **Operator → 설치된 Operator** 페이지로 이동합니다.
2. **Compliance Operator**가 **openshift-compliance** 네임스페이스에 설치되어 있고 해당 상태는 **Succeeded**인지 확인합니다.

Operator가 성공적으로 설치되지 않은 경우 다음을 수행하십시오.

1. **Operator → 설치된 Operator** 페이지로 이동하여 **Status** 열에 오류 또는 실패가 있는지 점검합니다.
2.

워크로드 → Pod 페이지로 이동하고 openshift-compliance 프로젝트에서 문제를 보고하는 Pod의 로그를 확인합니다.

5.3.2. CLI를 사용하여 Compliance Operator 설치

사전 요구 사항

- admin 권한이 있어야 합니다.

프로세스

1. Namespace 오브젝트를 정의합니다.

namespace-object.yaml의 예

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-compliance
```

2. Namespace 오브젝트를 생성합니다.

```
$ oc create -f namespace-object.yaml
```

3. OperatorGroup 오브젝트를 정의합니다.

예: operator-group-object.yaml

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: compliance-operator
  namespace: openshift-compliance
```

```
spec:
  targetNamespaces:
  - openshift-compliance
```

4. **OperatorGroup** 개체를 생성합니다.

```
$ oc create -f operator-group-object.yaml
```

5. **Subscription** 오브젝트를 정의합니다.

예: subscription-object.yaml

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "release-0.1"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

6. **Subscription** 오브젝트를 생성합니다.

```
$ oc create -f subscription-object.yaml
```

참고

글로벌 스케줄러 기능을 설정하고 **defaultNodeSelector**를 활성화하는 경우 네임스페이스를 수동으로 생성하고 **openshift-compliance** 네임스페이스의 주석 또는 **Compliance Operator**가 설치된 네임스페이스를 **openshift.io/node-selector: “”**로 업데이트해야 합니다. 이렇게 하면 기본 노드 선택기가 제거되고 배포 실패가 발생하지 않습니다.

검증

1. CSV 파일을 검사하여 설치에 성공했는지 확인합니다.

```
$ oc get csv -n openshift-compliance
```

2. Compliance Operator가 실행 중인지 확인합니다.

```
$ oc get deploy -n openshift-compliance
```

5.3.3. 추가 리소스

- Compliance Operator는 제한된 네트워크 환경에서 지원됩니다. 자세한 내용은 제한된 네트워크에서 Operator Lifecycle Manager 사용을 참조하십시오.

5.4. COMPLIANCE OPERATOR 검사

Compliance Operator를 사용하여 규정 준수 검사를 실행하도록 ScanSetting 및 ScanSettingBinding API를 사용하는 것이 좋습니다. 이러한 API 오브젝트에 대한 자세한 내용을 보려면 다음을 실행합니다.

```
$ oc explain scansettings
```

또는

```
$ oc explain scansettingbindings
```

5.4.1. 규정 준수 검사 실행

CIS(Center for Internet Security) 프로필을 사용하여 검사를 실행할 수 있습니다. 편의를 위해 Compliance Operator는 시작 시 적절한 기본값을 사용하여 ScanSetting 오브젝트를 생성합니다. 이 ScanSetting 오브젝트의 이름은 default 입니다.

참고

올인원 컨트롤 플레인 및 작업자 노드의 경우 규정 준수 스캔은 작업자 및 컨트롤 플레인 노드에서 두 번 실행됩니다. 규정 준수 검사에서 일관되지 않은 검사 결과를 생성할 수 있습니다. **ScanSetting** 오브젝트에서 단일 역할만 정의하여 일관성 없는 결과를 방지할 수 있습니다.

절차

1. 다음을 실행하여 **ScanSetting** 오브젝트를 검사합니다.

```
$ oc describe scansettings default -n openshift-compliance
```

출력 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  pvAccessModes:
  - ReadWriteOnce 1
  rotation: 3 2
  size: 1Gi 3
roles:
- worker 4
- master 5
scanTolerations: 6
  default:
  - operator: Exists
  schedule: 0 1 * * * 7
```

1. **Compliance Operator**는 검사 결과가 포함된 **PV**(영구 볼륨)를 생성합니다. 기본적으로 **PV**는 **Compliance Operator**에서 클러스터에 구성된 스토리지 클래스에 대한 가정을 할 수 없기 때문에 액세스 모드 **ReadWriteOnce**를 사용합니다. 대부분의 클러스터에서 **ReadWriteOnce** 액세스 모드를 사용할 수 있습니다. 검사 결과를 가져오려면 볼륨을 바인딩하는 도우미 **Pod**를 사용하여 이를 수행할 수 있습니다. **ReadWriteOnce** 액세스 모드를 사용하는 볼륨은 한 번에 하나의 **Pod**에서만 마운트할 수 있으므로 도우미 **Pod**를 삭제해야 합니다. 그러지 않으면 **Compliance Operator**가 후속 검사에 볼륨을 재사용할 수 없습니다.

2

Compliance Operator는 세 번의 후속 검사 결과를 볼륨에 보관합니다. 이전 검사는 순환됩니다.

3

Compliance Operator는 검사 결과에 대해 1GB의 스토리지를 할당합니다.

4 5

검사 설정에서 클러스터 노드를 검사하는 프로필을 사용하는 경우 이러한 노드 역할을 검사합니다.

6

기본 검사 설정 오브젝트도 모든 노드를 검사합니다.

7

기본 검사 설정 오브젝트는 매일 01:00에 검사를 실행합니다.

기본 검사 설정 대신 다음과 같은 설정이 있는 default-auto-apply를 사용할 수 있습니다.

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default-auto-apply
  namespace: openshift-compliance
autoUpdateRemediations: true 1
autoApplyRemediations: true 2
rawResultStorage:
  pvAccessModes:
    - ReadWriteOnce
  rotation: 3
  size: 1Gi
schedule: 0 1 * * *
roles:
  - worker
  - master
scanTolerations:
  default:
  - operator: Exists
```

1 2

autoUpdateRemediations 및 autoApplyRemediations 플래그를 true로 설정하면 추가 단계 없이 자동으로 조정되는 ScanSetting 오브젝트를 쉽게 생성할 수 있습니다.

2. 기본 ScanSetting 오브젝트에 바인딩하는 ScanSettingBinding 오브젝트를 생성하고 cis 및 cis-node 프로파일을 사용하여 클러스터를 검사합니다. 예를 들면 다음과 같습니다.

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis-compliance
  namespace: openshift-compliance
profiles:
  - name: ocp4-cis-node
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
  - name: ocp4-cis
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef:
  name: default
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

3. 다음을 실행하여 ScanSettingBinding 오브젝트를 생성합니다.

```
$ oc create -f <file-name>.yaml -n openshift-compliance
```

프로세스의 이 시점에서 ScanSettingBinding 오브젝트는 Binding 및 Bound 설정을 기반으로 조정됩니다. Compliance Operator는 ComplianceSuite 오브젝트 및 관련 ComplianceScan 오브젝트를 생성합니다.

4. 다음을 실행하여 컴플라이언스 검사 진행 상황을 따르십시오.

```
$ oc get compliancescan -w -n openshift-compliance
```

검사는 스캔 단계를 통해 진행되며 완료되면 DONE 단계에 도달합니다. 대부분의 경우 검사 결과는 NON-COMPLIANT입니다. 검사 결과를 검토하고 업데이트 적용 작업을 시작하여 클러스터를 준수하도록 할 수 있습니다. 자세한 내용은 Compliance Operator 업데이트 적용 관리를 참조하십시오.

5.4.2. 작업자 노드에 결과 서버 Pod 예약

결과 서버 포드는 원시 자산 보고 형식(ARF)을 저장하는 PV(영구 볼륨)를 마운트합니다. nodeSelector 및 tolerations 특성을 사용하면 결과 서버 포드의 위치를 구성할 수 있습니다.

이는 컨트롤 플레인 노드가 영구 볼륨을 마운트할 수 없는 환경에 유용합니다.

프로세스

- Compliance Operator의 ScanSetting CR(사용자 정의 리소스)을 생성합니다.

a. ScanSetting CR을 정의하고 YAML 파일을 저장합니다(예: rs-workers.yaml):

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: rs-on-workers
  namespace: openshift-compliance
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: "" (1)
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists (2)
roles:
- worker
- master
scanTolerations:
  - operator: Exists
schedule: 0 1 * * *
```

(1) Compliance Operator는 이 노드를 사용하여 검사 결과를 ARF 형식으로 저장합니다.

(2) 결과 서버 Pod는 모든 테인트를 허용합니다.

b. ScanSetting CR을 생성하려면 다음 명령을 실행합니다.

```
$ oc create -f rs-workers.yaml
```

검증

- ScanSetting 오브젝트가 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get scansettings rs-on-workers -n openshift-compliance -o yaml
```

출력 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  creationTimestamp: "2021-11-19T19:36:36Z"
  generation: 1
  name: rs-on-workers
  namespace: openshift-compliance
  resourceVersion: "48305"
  uid: 43fdfc5f-15a7-445a-8bbc-0e4a160cd46e
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists
roles:
- worker
- master
scanTolerations:
- operator: Exists
schedule: 0 1 * * *
strictNodeScan: true
```

5.5. COMPLIANCE OPERATOR 이해

OpenShift Container Platform 관리자는 Compliance Operator를 통해 클러스터의 필수 규정 준수 상태를 설명하고 격차에 대한 개요와 문제를 해결하는 방법을 제공할 수 있습니다. Compliance Operator는 OpenShift Container Platform의 Kubernetes API 리소스와 클러스터를 실행하는 노드 모두의 규정 준수를 평가합니다. Compliance Operator는 NIST 인증 툴인 OpenSCAP을 사용하여 콘텐츠에서 제공하는 보안 정책을 검사하고 시행합니다.

중요

Compliance Operator는 **Red Hat Enterprise Linux CoreOS (RHCOS)** 배포에만 사용할 수 있습니다.

5.5.1. Compliance Operator 프로필

Compliance Operator 설치의 일부로 다양한 프로필을 사용할 수 있습니다. **oc get** 명령을 사용하여 사용 가능한 프로필, 프로필 세부 정보 및 특정 규칙을 볼 수 있습니다.

- 사용 가능한 프로필 보기:

```
$ oc get -n <namespace> profiles.compliance
```

이 예에서는 기본 **openshift-compliance** 네임스페이스에 프로필을 표시합니다.

```
$ oc get -n openshift-compliance profiles.compliance
```

출력 예

```
NAME              AGE
ocp4-cis          32m
ocp4-cis-node     32m
ocp4-e8           32m
ocp4-moderate     32m
ocp4-moderate-node  32m
ocp4-nerc-cip     32m
ocp4-nerc-cip-node  32m
ocp4-pci-dss      32m
ocp4-pci-dss-node   32m
rhcos4-e8         32m
rhcos4-moderate   32m
rhcos4-nerc-cip   32m
```

이러한 프로필은 다양한 규정 준수 벤치마크를 나타냅니다. 각 프로필에는 적용되는 제품 이름이 프로필 이름에 접두사로 추가됩니다. **ocp4-e8** 은 **Essential 8** 벤치마크를 **OpenShift Container Platform** 제품에 적용하고, **rhcos4-e8** 은 **Essential 8** 벤치마크를 **RHCOS(Red Hat Enterprise Linux CoreOS)** 제품에 적용합니다.

- 프로필 세부 정보 보기:

```
$ oc get -n <namespace> -oyaml profiles.compliance <profile name>
```

이 예제에서는 **rhcos4-e8** 프로필의 세부 정보를 표시합니다.

```
$ oc get -n openshift-compliance -oyaml profiles.compliance rhcos4-e8
```

출력 예

```
apiVersion: compliance.openshift.io/v1alpha1
description: |-
  This profile contains configuration checks for Red Hat
  Enterprise Linux CoreOS that align to the Australian
  Cyber Security Centre (ACSC) Essential Eight.
  A copy of the Essential Eight in Linux Environments guide can
  be found at the ACSC website: ...
  id: xccdf_org.ssgproject.content_profile_e8
  kind: Profile
  metadata:
    annotations:
      compliance.openshift.io/image-digest: pb-rhcos426smj
      compliance.openshift.io/product: redhat_enterprise_linux_coreos_4
      compliance.openshift.io/product-type: Node
    labels:
      compliance.openshift.io/profile-bundle: rhcos4
    name: rhcos4-e8
    namespace: openshift-compliance
    ownerReferences:
    - apiVersion: compliance.openshift.io/v1alpha1
      blockOwnerDeletion: true
      controller: true
      kind: ProfileBundle
      name: rhcos4
  rules:
  - rhcos4-accounts-no-uid-except-zero
  - rhcos4-audit-rules-dac-modification-chmod
  - rhcos4-audit-rules-dac-modification-chown
  - rhcos4-audit-rules-execution-chcon
  - rhcos4-audit-rules-execution-restorecon
  - rhcos4-audit-rules-execution-semanage
  - rhcos4-audit-rules-execution-setfiles
  - rhcos4-audit-rules-execution-setsebool
  - rhcos4-audit-rules-execution-seunshare
  - rhcos4-audit-rules-kernel-module-loading-delete
  - rhcos4-audit-rules-kernel-module-loading-finit
  - rhcos4-audit-rules-kernel-module-loading-init
  - rhcos4-audit-rules-login-events
```

```
- rhcos4-audit-rules-login-events-faillock
- rhcos4-audit-rules-login-events-lastlog
- rhcos4-audit-rules-login-events-tallylog
- rhcos4-audit-rules-networkconfig-modification
- rhcos4-audit-rules-sysadmin-actions
- rhcos4-audit-rules-time-adjtimex
- rhcos4-audit-rules-time-clock-settime
- rhcos4-audit-rules-time-settimeofday
- rhcos4-audit-rules-time-stime
- rhcos4-audit-rules-time-watch-localtime
- rhcos4-audit-rules-usergroup-modification
- rhcos4-auditd-data-retention-flush
- rhcos4-auditd-freq
- rhcos4-auditd-local-events
- rhcos4-auditd-log-format
- rhcos4-auditd-name-format
- rhcos4-auditd-write-logs
- rhcos4-configure-crypto-policy
- rhcos4-configure-ssh-crypto-policy
- rhcos4-no-empty-passwords
- rhcos4-selinux-policytype
- rhcos4-selinux-state
- rhcos4-service-auditd-enabled
- rhcos4-sshd-disable-empty-passwords
- rhcos4-sshd-disable-gssapi-auth
- rhcos4-sshd-disable-rhosts
- rhcos4-sshd-disable-root-login
- rhcos4-sshd-disable-user-known-hosts
- rhcos4-sshd-do-not-permit-user-env
- rhcos4-sshd-enable-strictmodes
- rhcos4-sshd-print-last-log
- rhcos4-sshd-set-loglevel-info
- rhcos4-sysctl-kernel-dmesg-restrict
- rhcos4-sysctl-kernel-kptr-restrict
- rhcos4-sysctl-kernel-randomize-va-space
- rhcos4-sysctl-kernel-unprivileged-bpf-disabled
- rhcos4-sysctl-kernel-yama-ptrace-scope
- rhcos4-sysctl-net-core-bpf-jit-harden
title: Australian Cyber Security Centre (ACSC) Essential Eight
```

- 원하는 프로필 내 규칙 보기:

```
$ oc get -n <namespace> -oyaml rules.compliance <rule_name>
```

이 예에서는 **rhcos4** 프로필에 **rhcos4-audit-rules-login-events** 규칙을 표시합니다.

```
$ oc get -n openshift-compliance -oyaml rules.compliance rhcos4-audit-rules-login-events
```

-

출력 예

```
  apiVersion: compliance.openshift.io/v1alpha1
  checkType: Node
  description: |-
    The audit system already collects login information for all users and root. If the
auditd daemon is configured to use the augenrules program to read audit rules during
daemon startup (the default), add the following lines to a file with suffix.rules in the
directory /etc/audit/rules.d in order to watch for attempted manual edits of files
involved in storing logon events:

    -w /var/log/tallylog -p wa -k logins
    -w /var/run/faillock -p wa -k logins
    -w /var/log/lastlog -p wa -k logins

    If the auditd daemon is configured to use the auditctl utility to read audit rules
during daemon startup, add the following lines to /etc/audit/audit.rules file in order to
watch for unattempted manual edits of files involved in storing logon events:

    -w /var/log/tallylog -p wa -k logins
    -w /var/run/faillock -p wa -k logins
    -w /var/log/lastlog -p wa -k logins
  id: xccdf_org.ssgproject.content_rule_audit_rules_login_events
  kind: Rule
  metadata:
    annotations:
      compliance.openshift.io/image-digest: pb-rhcos426smj
      compliance.openshift.io/rule: audit-rules-login-events
      control.compliance.openshift.io/NIST-800-53: AU-2(d);AU-12(c);AC-6(9);CM-6(a)
      control.compliance.openshift.io/PCI-DSS: Req-10.2.3
      policies.open-cluster-management.io/controls: AU-2(d),AU-12(c),AC-6(9),CM-
6(a),Req-10.2.3
      policies.open-cluster-management.io/standards: NIST-800-53,PCI-DSS
    labels:
      compliance.openshift.io/profile-bundle: rhcos4
    name: rhcos4-audit-rules-login-events
    namespace: openshift-compliance
    ownerReferences:
    - apiVersion: compliance.openshift.io/v1alpha1
      blockOwnerDeletion: true
      controller: true
      kind: ProfileBundle
      name: rhcos4
  rationale: Manual editing of these files may indicate nefarious activity, such as
    an attacker attempting to remove evidence of an intrusion.
  severity: medium
  title: Record Attempts to Alter Logon and Logout Events
  warning: Manual editing of these files may indicate nefarious activity, such as an
attacker attempting to remove evidence of an intrusion.
```

5.6. COMPLIANCE OPERATOR 관리

이 섹션에서는 업데이트된 버전의 규정 준수 콘텐츠를 사용하는 방법과 사용자 정의 **ProfileBundle** 오브젝트를 만드는 방법을 포함하여 보안 콘텐츠의 라이프사이클에 대해 설명합니다.

5.6.1. 보안 콘텐츠 업데이트

보안 콘텐츠는 **ProfileBundle** 오브젝트에서 참조하는 컨테이너 이미지로 제공됩니다. **ProfileBundles** 및 규칙 또는 프로필과 같은 번들에서 구문 분석한 사용자 정의 리소스에 대한 업데이트를 정확하게 추적하려면 태그 대신 다이제스트를 사용하여 규정 준수 콘텐츠가 있는 컨테이너 이미지를 확인하십시오.

출력 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  name: rhcos4
spec:
  contentImage: quay.io/user/ocp4-openscap-
content@sha256:a1749f5150b19a9560a5732fe48a89f07bffc79c0832aa8c49ee5504590ae687 1
  contentFile: ssg-rhcos4-ds.xml
```

1

보안 컨테이너 이미지입니다.

각 **ProfileBundle**은 배포를 통해 지원됩니다. **Compliance Operator**에서 컨테이너 이미지 다이제스트가 변경되었음을 감지하면 배포가 업데이트되어 변경 사항을 반영하고 콘텐츠를 다시 구문 분석합니다. 태그 대신 다이제스트를 사용하면 안정적이고 예측 가능한 프로필 세트를 사용할 수 있습니다.

5.6.2. 이미지 스트림 사용

contentImage 참조가 유효한 **ImageStreamTag**를 가리키고 **Compliance Operator**를 통해 콘텐츠를 자동으로 최신 상태로 유지합니다.

참고

ProfileBundle 오브젝트에는 **ImageStream** 참조도 사용할 수 있습니다.

이미지 스트림 예

```
$ oc get is -n openshift-compliance
```

출력 예

```
NAME              IMAGE REPOSITORY                                                                TAGS
UPDATED
openscap-ocp4-ds   image-registry.openshift-image-registry.svc:5000/openshift-
compliance/openscap-ocp4-ds   latest   32 seconds ago
```

프로세스

1. 조회 정책이 로컬로 설정되어 있는지 확인합니다.

```
$ oc patch is openscap-ocp4-ds \
  -p '{"spec":{"lookupPolicy":{"local":true}}}' \
  --type=merge
  imagestream.image.openshift.io/openscap-ocp4-ds patched
  -n openshift-compliance
```

2. **istag** 이름을 검색하여 **ProfileBundle**에 **ImageStreamTag** 이름을 사용합니다.

```
$ oc get istag -n openshift-compliance
```

출력 예

```
NAME                 IMAGE REFERENCE
UPDATED
openscap-ocp4-ds:latest   image-registry.openshift-image-registry.svc:5000/openshift-
compliance/openscap-ocp4-
ds@sha256:46d7ca9b7055fe56ade818ec3e62882cfcc2d27b9bf0d1cbae9f4b6df2710c96
3 minutes ago
```

3. ProfileBundle을 생성합니다.

```
$ cat << EOF | oc create -f -
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  name: mybundle
   spec:
     contentImage: openscap-ocp4-ds:latest
     contentFile: ssg-rhcos4-ds.xml
EOF
```

이 ProfileBundle은 해당 이미지를 추적하고 다른 해시를 가리키도록 태그를 업데이트하는 등 이 오브젝트에 적용되는 모든 변경 사항은 ProfileBundle에 즉시 반영됩니다.

5.6.3. ProfileBundle CR의 예

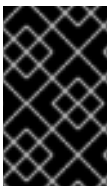
번들 오브젝트에는 contentImage가 포함된 컨테이너 이미지의 URL과 규정 준수 콘텐츠가 포함된 파일 정보가 필요합니다. contentFile 매개변수는 파일 시스템의 루트와 상대적입니다. 내장된 rhcos4 ProfileBundle 오브젝트는 아래 예제에서 정의할 수 있습니다.

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  name: rhcos4
spec:
  contentImage: quay.io/complianceascode/ocp4:latest 1
  contentFile: ssg-rhcos4-ds.xml 2
```

1 콘텐츠 이미지 위치입니다.

2

규정 준수 콘텐츠가 포함된 파일의 위치입니다.



중요

콘텐츠 이미지에 사용되는 기본 이미지에는 **coreutils**가 포함되어야 합니다.

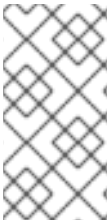
5.6.4. 추가 리소스

- **Compliance Operator**는 제한된 네트워크 환경에서 지원됩니다. 자세한 내용은 제한된 네트워크에서 **Operator Lifecycle Manager** 사용을 참조하십시오.

5.7. COMPLIANCE OPERATOR 조정

Compliance Operator는 즉시 사용할 수 있는 프로필과 함께 제공되지만 조직의 필요 및 요구 사항에 맞게 수정해야 합니다. 프로필을 수정하는 프로세스를 *조정*이라고 합니다.

Compliance Operator에서는 **TailoredProfile**이라는 프로필을 쉽게 조정할 수 있는 오브젝트를 제공합니다. 이 프로필은 사용자가 기존 프로필을 확장하는 것으로 가정하고 **ProfileBundle**에서 제공하는 규칙과 값을 활성화 및 비활성화할 수 있도록 허용합니다.



참고

확장하려는 프로필이 속한 **ProfileBundle**의 일부로 사용할 수 있는 규칙과 변수만 사용할 수 있습니다.

5.7.1. 맞춤형 프로필 사용

TailoredProfile CR에서는 가장 일반적인 맞춤 작업을 수행할 수 있지만 **XCCDF** 표준을 사용하면 **OpenSCAP** 프로필 맞춤 시 유연성이 훨씬 더 향상됩니다. 또한 조직에서 이전에 **OpenScap**을 사용한 적이 있는 경우 기존 **XCCDF** 맞춤 파일이 있을 수 있으며 이 파일을 다시 사용할 수 있습니다.

ComplianceSuite 오브젝트에는 사용자 정의 맞춤 파일을 가리킬 수 있는 선택적 **TailoringConfigMap** 특성이 포함되어 있습니다. **TailoringConfigMap** 특성 값은 구성 맵의 이름으로, 이 맵에는 **tailoring.xml**이라는 키가 포함되어야 하며 이 키의 값은 맞춤 콘텐츠입니다.

프로세스

1. RHCOS(Red Hat Enterprise Linux CoreOS) ProfileBundle에 사용 가능한 규칙을 찾습니다.

```
$ oc get rules.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

2. 해당 ProfileBundle에서 사용 가능한 변수를 찾습니다.

```
$ oc get variables.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

3. 이름이 nist-moderate-modified 인 맞춤형 프로필을 생성합니다.

a. nist-moderate-modified tailored profile에 추가할 규칙을 선택합니다. 이 예제에서는 두 개의 규칙을 비활성화하고 하나의 값을 변경하여 rhcos4-moderate 프로필을 확장합니다. rationale 값을 사용하여 이러한 변경이 이루어진 이유를 설명합니다.

예: new-profile-node.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: nist-moderate-modified
spec:
  extends: rhcos4-moderate
  description: NIST moderate profile
  title: My modified NIST moderate profile
  disableRules:
  - name: rhcos4-file-permissions-var-log-messages
    rationale: The file contains logs of error messages in the system
  - name: rhcos4-account-disable-post-pw-expiration
    rationale: No need to check this as it comes from the IdP
  setValues:
  - name: rhcos4-var-selinux-state
    rationale: Organizational requirements
    value: permissive
```

표 5.2. spec 변수의 속성

속성	설명
extends	이 **TailoredProfile**이 빌드되는 **Profile** 오브젝트의 이름입니다.
title	**TailoredProfile**의 사람이 읽을 수 있는 제목입니다.
disableRules	이름 및 이유 쌍 목록입니다. 각 이름은 비활성화할 규칙 오브젝트의 이름을 나타냅니다. 이유 값은 규칙이 비활성화된 이유를 설명하는 사람이 읽을 수 있는 텍스트입니다.
enableRules	이름 및 이유 쌍 목록입니다. 각 이름은 활성화할 규칙 오브젝트의 이름을 나타냅니다. 이유 값은 규칙이 활성화된 이유를 설명하는 사람이 읽을 수 있는 텍스트입니다.
description	**TailoredProfile**을 설명하는 사람이 읽을 수 있는 텍스트입니다.
setValues	이름, 이유 및 값 그룹화 목록입니다. 각 이름은 설정된 값의 이름을 나타냅니다. 이유는 집합을 설명하는 사람이 읽을 수 있는 텍스트입니다. 값은 실제 설정입니다.

b. **TailoredProfile 오브젝트를 생성합니다.**

```
$ oc create -n openshift-compliance -f new-profile-node.yaml 1
```

1 **TailoredProfile 오브젝트는 기본 openshift-compliance 네임스페이스에 생성됩니다.**

출력 예

```
tailoredprofile.compliance.openshift.io/nist-moderate-modified created
```

4. **새 nist-moderate-modified tailored 프로필을 기본 ScanSetting 오브젝트에 바인딩하도록 ScanSettingBinding 오브젝트를 정의합니다.**

new-scansettingbinding.yaml의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: nist-moderate-modified
profiles:
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: Profile
    name: ocp4-moderate
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: TailoredProfile
    name: nist-moderate-modified
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default
```

5. **ScanSettingBinding** 오브젝트를 생성합니다.

```
$ oc create -n openshift-compliance -f new-scansettingbinding.yaml
```

출력 예

```
scansettingbinding.compliance.openshift.io/nist-moderate-modified created
```

5.8. COMPLIANCE OPERATOR 원시 결과 검색

OpenShift Container Platform 클러스터의 규정 준수를 입증할 때 감사 목적으로 검사 결과를 제공해야 할 수 있습니다.

5.8.1. 영구 볼륨에서 Compliance Operator 원시 결과 가져오기

프로세스

Compliance Operator는 원시 결과를 생성하여 영구 볼륨에 저장합니다. 이러한 결과는 자산 보고 형식**(ARF)**으로 되어 있습니다.

1. **ComplianceSuite** 오브젝트를 살펴봅니다.

```
$ oc get compliancesuites nist-moderate-modified -o json \
  | jq '.status.scanStatuses[].resultsStorage'
  {
    "name": "rhcos4-moderate-worker",
    "namespace": "openshift-compliance"
  }
  {
    "name": "rhcos4-moderate-master",
    "namespace": "openshift-compliance"
  }
```

원시 결과에 액세스할 수 있는 영구 볼륨 클레임이 표시됩니다.

2. 결과 중 하나의 이름과 네임스페이스를 사용하여 원시 데이터 위치를 확인합니다.

```
$ oc get pvc -n openshift-compliance rhcos4-moderate-worker
```

출력 예

```
NAME                    STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
rhcos4-moderate-worker   Bound pvc-548f6cfe-164b-42fe-ba13-a07cfbc77f3a   1Gi
RWO        gp2        92m
```

3. 볼륨을 마운트하는 **Pod**를 생성하고 결과를 복사하여 원시 결과를 가져옵니다.

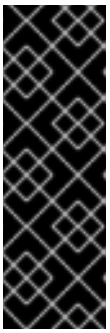
Pod 예

```
apiVersion: "v1"
kind: Pod
metadata:
  name: pv-extract
spec:
  containers:
    - name: pv-extract-pod
```

```
    image: registry.access.redhat.com/ubi8/ubi
    command: ["sleep", "3000"]
    volumeMounts:
    - mountPath: "/workers-scan-results"
      name: workers-scan-vol
volumes:
  - name: workers-scan-vol
    persistentVolumeClaim:
      claimName: rhcos4-moderate-worker
```

4. Pod가 실행되면 결과를 다운로드합니다.

```
$ oc cp pv-extract:/workers-scan-results .
```



중요

영구 볼륨을 마운트하는 Pod를 생성하면 클레임이 Bound로 유지됩니다. 사용 중인 볼륨의 스토리지 클래스에 ReadWriteOnce로 설정된 권한이 있는 경우 한 번에 하나의 Pod에서만 볼륨을 마운트할 수 있습니다. 완료 시 Pod를 삭제해야 합니다. 그러지 않으면 Operator에서 Pod를 예약하고 이 위치에 결과를 계속 저장할 수 없습니다.

5. 추출이 완료되면 Pod를 삭제할 수 있습니다.

```
$ oc delete pod pv-extract
```

5.9. COMPLIANCE OPERATOR 결과 및 수정 관리

각 ComplianceCheckResult는 하나의 규정 준수 규칙 검사 결과를 나타냅니다. 규칙을 자동으로 수정할 수 있는 경우 ComplianceCheckResult가 소유한 동일한 이름의 ComplianceRemediation 오브젝트가 생성됩니다. 요청하지 않는 경우 수정은 자동으로 적용되지 않으므로 OpenShift Container Platform 관리자는 수정을 통해 수행되는 작업을 검토하고 확인한 후에만 수정을 적용할 수 있습니다.

5.9.1. 규정 준수 점검 결과에 대한 필터

기본적으로 ComplianceCheckResult 오브젝트에는 검사를 쿼리하고 결과가 생성된 후 다음 단계를 결정할 수 있는 몇 가지 유용한 레이블이 지정됩니다.

특정 제품군에 속하는 검사를 나열합니다.

```
$ oc get compliancecheckresults -l compliance.openshift.io/suite=example-compliancesuite
```

특정 검사에 속하는 검사를 나열합니다.

```
$ oc get compliancecheckresults -l compliance.openshift.io/scan=example-compliancescan
```

일부 ComplianceCheckResult 오브젝트가 ComplianceRemediation 오브젝트를 생성하는 것은 아닙니다. 자동으로 업데이트를 적용할 수 있는 ComplianceCheckResult 오브젝트만 해당합니다. ComplianceCheckResult 오브젝트에 compliance.openshift.io/automated-remediation 레이블이 지정된 경우 관련된 업데이트 적용이 있습니다. 업데이트 적용의 이름은 검사 이름과 동일합니다.

자동으로 업데이트를 적용할 수 있는 모든 실패한 검사를 나열합니다.

```
$ oc get compliancecheckresults -l 'compliance.openshift.io/check-
status=FAIL,compliance.openshift.io/automated-remediation'
```

수동으로 업데이트를 적용해야 하는 모든 실패한 검사를 나열합니다.

```
$ oc get compliancecheckresults -l 'compliance.openshift.io/check-
status=FAIL,!compliance.openshift.io/automated-remediation'
```

수동 업데이트 적용 단계는 일반적으로 ComplianceCheckResult 오브젝트의 description 속성에 저장됩니다.

표 5.3. ComplianceCheckResult 상태

ComplianceCheckResult 상태	설명
PASS	규정 준수 점검이 완료 및 통과되었습니다.
실패	규정 준수 검사가 완료되었으며 실패했습니다.
정보	컴플라이언스 검사가 완료되고 오류로 간주될 만큼 심각하지 않은 것을 발견했습니다.
MANUAL	컴플라이언스 확인에는 성공 또는 실패를 자동으로 평가할 방법이 없으며 수동으로 확인해야 합니다.

ComplianceCheckResult 상태	설명
일관되지 않음	컴플라이언스 점검은 다른 소스(일반적으로 클러스터 노드)의 다른 결과를 보고합니다.
오류	규정 준수 검사가 실행되었지만 제대로 완료할 수 없습니다.
적용되지 않음	규정 준수 점검은 적용되지 않거나 선택되지 않았기 때문에 실행되지 않았습니다.

5.9.2. 수정 검토

수정이 포함된 **ComplianceRemediation** 오브젝트 및 **ComplianceCheckResult** 오브젝트를 모두 검토합니다. **ComplianceCheckResult** 오브젝트에는 검사에서 수행하는 작업과 방지를 위한 강화 작업에 관해 사람이 있을 수 있는 설명과 심각도 및 관련 보안 제어와 같은 기타 메타데이터가 포함되어 있습니다. **ComplianceRemediation** 오브젝트는 **ComplianceCheckResult**에서 설명하는 문제를 해결하는 방법을 나타냅니다. 첫 번째 검사 후 상태가 누락된 상태로 수정되는지 확인합니다.

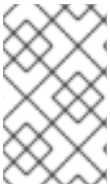
다음은 **sysctl-net-ipv4-conf-all-accept-redirects**라는 검사와 수정의 예입니다. 이 예는 **spec** 및 **status**만 표시하고 **metadata**는 생략하도록 수정되었습니다.

```
spec:
  apply: false
  current:
  object:
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
    spec:
      config:
        ignition:
          version: 3.1.0
        storage:
          files:
            - path: /etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf
              mode: 0644
              contents:
                source: data:,net.ipv4.conf.all.accept_redirects%3D0
  outdated: {}
status:
  applicationState: NotApplied
```

수정 페이로드는 **spec.current** 특성에 저장됩니다. 페이로드는 임의의 **Kubernetes** 오브젝트일 수 있지만 이 수정은 노드 검사를 통해 생성되었기 때문에 위 예의 수정 페이로드는 **MachineConfig** 오브젝트입니다. 플랫폼 검사의 경우 수정 페이로드는 종종 다른 종류의 오브젝트(예: **ConfigMap** 또는 **Secret** 오브젝트)에 해당하지만 일반적으로 이러한 수정을 적용하는 것은 관리자의 몫입니다. 그러지 않으면 일반

Kubernetes 오브젝트를 조작하기 위해 **Compliance Operator**에 매우 광범위한 권한이 있어야 하기 때문입니다. 플랫폼 검사를 수정하는 예는 본문 뒷부분에 있습니다.

수정 적용 시 수행되는 작업을 정확히 확인하기 위해 **MachineConfig** 오브젝트 콘텐츠에서는 구성에 **Ignition** 오브젝트를 사용합니다. 형식에 대한 자세한 내용은 **Ignition** 사양을 참조하십시오. 이 예에서 **spec.config.storage.files[0].path** 특성은 이 수정(**/etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf**)으로 생성되는 파일을 지정하고, **spec.config.storage.files[0].contents.source** 특성은 해당 파일의 콘텐츠를 지정합니다.



참고

파일 내용은 **URL**로 인코딩됩니다.

콘텐츠를 보려면 다음 **Python** 스크립트를 사용합니다.

```
$ echo "net.ipv4.conf.all.accept_redirects%3D0" | python3 -c "import sys, urllib.parse;
print(urllib.parse.unquote(".join(sys.stdin.readlines())))"
```

출력 예

```
net.ipv4.conf.all.accept_redirects=0
```

5.9.3. 사용자 지정 머신 구성 풀을 사용할 때 수정 사항 적용

사용자 지정 **MachineConfigPool** 을 생성할 때 **KubeletConfig** 에 있는 **machineConfigPoolSelector** 가 **MachineConfigPool**과 일치하는 라벨과 일치하도록 **MachineConfigPool** 에 레이블을 추가합니다.

중요

Compliance Operator가 수정 적용을 완료한 후 **MachineConfigPool** 오브젝트가 예기치 않게 일시 중지되지 않을 수 있으므로 **KubeletConfig** 파일에서 **protectKernelDefaults: false** 를 설정하지 마십시오.

프로세스

1. 노드를 나열합니다.

```
$ oc get nodes
```

출력 예

```
NAME                                        STATUS  ROLES  AGE    VERSION
ip-10-0-128-92.us-east-2.compute.internal  Ready   master 5h21m  v1.23.3+d99c04f
ip-10-0-158-32.us-east-2.compute.internal  Ready   worker 5h17m  v1.23.3+d99c04f
ip-10-0-166-81.us-east-2.compute.internal  Ready   worker 5h17m  v1.23.3+d99c04f
ip-10-0-171-170.us-east-2.compute.internal Ready   master 5h21m  v1.23.3+d99c04f
ip-10-0-197-35.us-east-2.compute.internal  Ready   master 5h22m  v1.23.3+d99c04f
```

2. 노드에 레이블을 추가합니다.

```
$ oc label node ip-10-0-166-81.us-east-2.compute.internal node-
role.kubernetes.io/<machine_config_pool_name>=
```

출력 예

```
node/ip-10-0-166-81.us-east-2.compute.internal labeled
```

3. 사용자 지정 **MachineConfigPool CR**을 생성합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: <machine_config_pool_name>
  labels:
    pools.operator.machineconfiguration.openshift.io/<machine_config_pool_name>: ''
1
spec:
  machineConfigSelector:
  matchExpressions:
  - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,
```

```
<machine_config_pool_name>]}
  nodeSelector:
  matchLabels:
    node-role.kubernetes.io/<machine_config_pool_name>: ""
```

1

labels 필드는 MCP(Machine config pool)에 추가할 레이블 이름을 정의합니다.

4. MCP가 성공적으로 생성되었는지 확인합니다.

```
$ oc get mcp -w
```

5.9.4. 수정 적용

부울 특성 spec.apply는 Compliance Operator에서 수정을 적용해야 하는지를 제어합니다. 특성을 true로 설정하면 수정을 적용할 수 있습니다.

```
$ oc patch complianceremediations/<scan_name>-sysctl-net-ipv4-conf-all-accept-redirects --
patch '{"spec":{"apply":true}}' --type=merge
```

Compliance Operator에서 적용된 수정을 처리하면 status.ApplicationState 특성이 Applied로 변경되거나 잘못된 경우 Error로 변경됩니다. 시스템 구성 수정이 적용되면 적용된 기타 모든 수정과 함께 해당 수정이 75-$scan-name-$suite-name이라는 MachineConfig 오브젝트로 렌더링됩니다. 이후 Machine Config Operator에서 MachineConfig 오브젝트를 렌더링하고 마지막으로 각 노드에서 실행되는 머신 제어 데몬 인스턴스에서 머신 구성 풀의 모든 노드에 이 오브젝트를 적용합니다.

Machine Config Operator에서 새 MachineConfig 오브젝트를 풀의 노드에 적용하면 풀에 속하는 모든 노드가 재부팅됩니다. 이러한 방법은 복합적인 75-$scan-name-$suite-name MachineConfig 오브젝트를 각각 다시 렌더링하는 수정을 여러 번 적용할 때 불편할 수 있습니다. 수정을 즉시 적용하지 않으려면 MachineConfigPool 오브젝트의 .spec.paused 특성을 true로 설정하여 머신 구성 풀을 일시 중지하면 됩니다.

Compliance Operator는 수정을 자동으로 적용할 수 있습니다. ScanSetting 최상위 오브젝트에 autoApplyRemediations: true를 설정합니다.

주의

수정 사항 자동 적용은 신중하게 고려해야 합니다.

5.9.5. 플랫폼 점검 수동 수정

플랫폼 검사에 대한 점검은 일반적으로 다음 두 가지 이유로 관리자가 수동으로 수정해야 합니다.

- 설정해야 하는 값을 자동으로 결정할 수 없는 경우가 있습니다. 검사 중 하나를 통해 허용된 레지스트리 목록을 제공해야 하지만 스캐너에서는 조직이 허용하려는 레지스트리를 알 수 없습니다.
- 다양한 점검에서 여러 **API** 오브젝트를 수정하므로 클러스터의 오브젝트를 수정하려면 **root** 또는 슈퍼 유저 액세스 권한을 가져오기 위해 자동 수정이 필요합니다. 이 방법은 바람직하지 않습니다.

프로세스

1. 아래 예제에서는 **ocp4-ocp-allowed-registries-for-import** 규칙을 사용하며 기본 **OpenShift Container Platform** 설치에서 실패합니다. **oc get rule.compliance/ocp4-ocp-allowed-registries-for-import -oyaml** 규칙을 검사합니다. 이 규칙은 **allowedRegistriesForImport** 특성을 설정하여 사용자가 이미지를 가져올 수 있는 레지스트리를 제한합니다. 규칙의 ***warning*** 특성에는 점검된 **API** 오브젝트도 표시되므로 이를 수정하고 문제를 해결할 수 있습니다.

```
$ oc edit image.config.openshift.io/cluster
```

출력 예

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2020-09-10T10:12:54Z"
  generation: 2
  name: cluster
  resourceVersion: "363096"
  selfLink: /apis/config.openshift.io/v1/images/cluster
```

```
  uid: 2dcb614e-2f8a-4a23-ba9a-8e33cd0ff77e
spec:
  allowedRegistriesForImport:
  - domainName: registry.redhat.io
status:
  externalRegistryHostnames:
  - default-route-openshift-image-registry.apps.user-cluster-09-10-12-
07.devcluster.openshift.com
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

2. 검사를 다시 실행합니다.

```
$ oc annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

5.9.6. 수정 업데이트

새 버전의 규정 준수 콘텐츠를 사용하는 경우 이전 버전과 다른 새 버전의 수정을 제공할 수 있습니다. Compliance Operator는 이전 버전의 수정을 적용한 상태로 유지됩니다. OpenShift Container Platform 관리자에게는 검토하고 적용할 새 버전에 대한 알림이 제공됩니다. 이전에 적용되었지만 업데이트된 ComplianceRemediation 오브젝트는 상태가 Outdated로 변경됩니다. 오래된 오브젝트는 쉽게 검색할 수 있도록 레이블이 지정됩니다.

이전에 적용된 수정 내용은 ComplianceRemediation 오브젝트의 spec.outdated 특성에 저장되고 새로 업데이트된 내용은 spec.current 특성에 저장됩니다. 콘텐츠가 최신 버전으로 업데이트되면 관리자는 수정을 검토해야 합니다. spec.outdated 특성이 존재하는 동안에는 결과 MachineConfig 오브젝트를 렌더링하는 데 사용됩니다. spec.outdated 특성이 제거되면 Compliance Operator에서 결과 MachineConfig 오브젝트를 다시 렌더링하고 이로 인해 Operator에서 구성을 노드로 푸시합니다.

프로세스

1. 오래된 수정을 검색합니다.

```
$ oc get complianceremediations -lcomplianceoperator.openshift.io/outdated-
remediation=
```

출력 예

```
NAME                          STATE
workers-scan-no-empty-passwords   Outdated
```

현재 적용된 수정은 **Outdated** 특성에 저장되고 적용되지 않은 새 수정은 **Current** 특성에 저장됩니다. 새 버전에 만족한다면 **Outdated** 필드를 제거하십시오. 업데이트된 콘텐츠를 유지하려면 **Current** 및 **Outdated** 특성을 제거하십시오.

2. 최신 버전의 수정을 적용합니다.

```
$ oc patch complianceremediations workers-scan-no-empty-passwords --type json -p
'[{"op":"remove", "path":/spec/outdated}]'
```

3. 수정 상태가 **Outdated**에서 **Applied**로 전환됩니다.

```
$ oc get complianceremediations workers-scan-no-empty-passwords
```

출력 예

```
NAME                           STATE
workers-scan-no-empty-passwords   Applied
```

4. 노드에 최신 수정 버전이 적용되고 노드가 재부팅됩니다.

5.9.7. 수정 적용 취소

이전에 적용한 수정을 적용 취소해야 할 수 있습니다.

프로세스

1. **apply** 플래그를 **false** 로 설정합니다. :

```
$ oc patch complianceremediations/<scan_name>-sysctl-net-ipv4-conf-all-accept-
redirects -p '{"spec":{"apply":false}}' --type=merge
```

2.

수정 상태가 **NotApplied**로 변경되고 복합 **MachineConfig** 오브젝트가 수정을 포함하지 않도록 다시 렌더링됩니다.

중요

수정으로 영향을 받는 모든 노드가 재부팅됩니다.

5.9.8. KubeletConfig 수정 제거

KubeletConfig 수정은 노드 수준 프로필에 포함됩니다. **KubeletConfig** 수정을 제거하려면 **KubeletConfig** 오브젝트에서 수동으로 제거해야 합니다. 이 예에서는 **one-rule-tp-node-master-kubelet-eviction-thresholds-hard-imagefs-available** 수정에 대한 규정 준수 검사를 제거하는 방법을 보여줍니다.

프로세스

1.

one-rule -tp-node-master-kubelet-eviction-thresholds-hard-imagefs-available 수정에 대한 **scan- name** 및 컴플라이언스 검사를 찾습니다.

```
$ oc get remediation one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available -o yaml
```

출력 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  annotations:
    compliance.openshift.io/xccdf-value-used: var-kubelet-evictionhard-imagefs-available
  creationTimestamp: "2022-01-05T19:52:27Z"
  generation: 1
  labels:
    compliance.openshift.io/scan-name: one-rule-tp-node-master (1)
    compliance.openshift.io/suite: one-rule-ssb-node
  name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
```

```
    name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-
available
    uid: fe8e1577-9060-4c59-95b2-3e2c51709adc
  resourceVersion: "84820"
  uid: 5339d21a-24d7-40cb-84d2-7a2ebb015355
spec:
  apply: true
  current:
    object:
      apiVersion: machineconfiguration.openshift.io/v1
      kind: KubeletConfig
      spec:
        kubeletConfig:
          evictionHard:
            imagefs.available: 10% 2
  outdated: {}
  type: Configuration
status:
  applicationState: Applied
```

1

수정의 검사 이름입니다.

2

KubeletConfig 오브젝트에 추가된 수정입니다.

참고

수정에서 **evictionHard kubelet** 구성을 호출하는 경우 **evictionHard** 매개변수인 **memory.available,nodefs.available,nodefs.inodesFree,imagefs.available, imagefs.inodesFree** 를 지정해야 합니다. 모든 매개변수를 지정하지 않으면 지정된 매개변수만 적용되고 수정이 제대로 작동하지 않습니다.

2. 수정을 제거합니다.

   a. 수정 오브젝트에 **apply** 를 **false**로 설정합니다.

```
$ oc patch complianceremediations/one-rule-tp-node-master-kubelet-eviction-
thresholds-set-hard-imagefs-available -p '{"spec":{"apply":false}}' --type=merge
```

b.

scan-name 을 사용하여 수정이 적용된 **KubeletConfig** 오브젝트를 찾습니다.

```
$ oc get kubeletconfig --selector compliance.openshift.io/scan-name=one-rule-tp-node-master
```

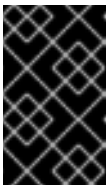
출력 예

```
NAME                            AGE
compliance-operator-kubelet-master   2m34s
```

c.

수정 **imagefs.available**을 수동으로 제거합니다. **KubeletConfig** 오브젝트에서 **10%:**

```
$ oc edit KubeletConfig compliance-operator-kubelet-master
```



중요

수정으로 영향을 받는 모든 노드가 재부팅됩니다.

참고

또한 수정 사항을 자동 적용하는 사용자 정의 프로필의 예약된 검사에서 규칙을 제외해야 합니다. 그렇지 않으면 다음 예정된 검사 중에 수정이 다시 적용됩니다.

5.9.9. 일관성 없는 ComplianceScan

ScanSetting 오브젝트는 **ScanSetting** 또는 **ScanSettingBinding** 오브젝트에서 생성한 규정 준수 검사에서 검사할 노드 역할을 나열합니다. 각 노드 역할은 일반적으로 머신 구성 풀에 매핑됩니다.

중요

머신 구성 풀의 모든 머신이 동일하고 풀에 있는 노드의 모든 검사 결과가 동일해야 합니다.

일부 결과가 다른 결과와 다른 경우 Compliance Operator는 일부 노드에서 INCONSISTENT로 보고하는 ComplianceCheckResult 오브젝트에 플래그를 지정합니다. 또한 모든 ComplianceCheckResult 오브젝트에는 compliance.openshift.io/inconsistent-check 레이블이 지정됩니다.

풀의 머신 수가 상당히 많을 수 있기 때문에 Compliance Operator는 가장 일반적인 상태를 찾고 일반적인 상태와 다른 노드를 나열하려고 합니다. 가장 일반적인 상태는 compliance.openshift.io/most-common-status 주석에 저장되고 주석 compliance.openshift.io/inconsistent-source에는 가장 일반적인 상태와 다른 점검 상태의 hostname:status 쌍이 포함됩니다. 일반적인 상태를 찾을 수 없는 경우 모든 hostname:status 쌍이 compliance.openshift.io/inconsistent-source annotation에 나열됩니다.

가능한 경우 클러스터가 규정 준수 상태에 통합될 수 있도록 수정이 계속 생성됩니다. 그러나 이러한 통합이 항상 가능한 것은 아니며 노드 간 차이를 수동으로 수정해야 합니다.
compliance.openshift.io/rescan= 옵션으로 검사에 주석을 달아 일관된 결과를 가져오도록 규정 준수 검사를 다시 실행해야 합니다.

```
$ oc annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

5.9.10. 추가 리소스

- 노드 수정.

5.10. 고급 COMPLIANCE OPERATOR 작업 수행

Compliance Operator에는 디버깅 또는 기존 툴과의 통합에 필요한 고급 사용자용 옵션이 포함되어 있습니다.

5.10.1. ComplianceSuite 및 ComplianceScan 오브젝트 직접 사용

사용자가 ScanSetting 및 ScanSettingBinding 오브젝트를 활용하여 모음과 검사를 정의하는 것이 바람직하지만 ComplianceSuite 오브젝트를 직접 정의하는 유효한 사용 사례가 있습니다.

- 검사할 단일 규칙만 지정합니다. 그러지 않으면 디버그 모드가 매우 상세하게 표시되는 경향이 있으므로 이 방법은 OpenSCAP 스캐너의 상세 수준을 높이는 debug: true 특성과 함께 디버깅하는 데 유용할 수 있습니다. 테스트를 하나의 규칙으로 제한하면 디버그 정보의 양을 줄이는 데 도움이 됩니다.

- 사용자 정의 nodeSelector를 제공합니다. 수정을 적용하려면 nodeSelector가 풀과 일치해야 합니다.

- 맞춤 파일을 사용하여 맞춤형 구성 맵을 검사합니다.

- 번들의 프로파일을 구문 분석하는 오버헤드가 필요하지 않은 경우의 테스트 또는 개발에 해당합니다.

다음 예제에서는 단일 규칙으로만 작업자 머신을 검사하는 **ComplianceSuite**를 보여줍니다.

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: quay.io/complianceascode/ocp4:latest
      debug: true
      rule: xccdf_org.ssgproject.content_rule_no_direct_root_logins
      nodeSelector:
      node-role.kubernetes.io/worker: ""
```

위에서 언급한 **ComplianceSuite** 오브젝트 및 **ComplianceScan** 오브젝트는 여러 특성을 **OpenSCAP**에서 예상하는 형식으로 지정합니다.

프로필, 콘텐츠 또는 규칙 값을 찾으려면 **ScanSetting** 및 **ScanSettingBinding**에서 유사한 모음을 생성하여 시작하거나 규칙 또는 프로필과 같이 **ProfileBundle** 오브젝트에서 구문 분석한 오브젝트를 검사하면 됩니다. 이러한 오브젝트에는 **ComplianceSuite**에서 참조하는 데 사용할 수 있는 **xccdf_org** 식별자가 포함되어 있습니다.

5.10.2. 원시 맞춤형 프로필 사용

TailoredProfile CR에서는 가장 일반적인 맞춤 작업을 수행할 수 있지만 **XCCDF** 표준을 사용하면 **OpenSCAP** 프로필 맞춤 시 유연성이 훨씬 더 향상됩니다. 또한 조직에서 이전에 **OpenScap**을 사용한 적이 있는 경우 기존 **XCCDF** 맞춤 파일이 있을 수 있으며 이 파일을 다시 사용할 수 있습니다.

ComplianceSuite 오브젝트에는 사용자 정의 맞춤 파일을 가리킬 수 있는 선택적 **TailoringConfigMap** 특성이 포함되어 있습니다. **TailoringConfigMap** 특성 값은 구성 맵의 이름으로, 이 맵에는 **tailoring.xml**이라는 키가 포함되어야 하며 이 키의 값은 맞춤 콘텐츠입니다.

프로세스

1. 파일에서 **ConfigMap** 오브젝트를 만듭니다.

```
$ oc create configmap <scan_name> --from-
file=tailoring.xml=/path/to/the/tailoringFile.xml
```

2. 모음에 속하는 검사의 맞춤 파일을 참조합니다.

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  debug: true
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: quay.io/complianceascode/ocp4:latest
      debug: true
  tailoringConfigMap:
      name: <scan_name>
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

5.10.3. 재검사 수행

일반적으로 매주 월요일 또는 매일 등 정의된 일정에 따라 검사를 다시 실행하려고 할 것입니다. 노드 문제를 해결한 후 다시 한 번 검사를 실행하는 것도 유용할 수 있습니다. 단일 검사를 수행하려면 **compliance.openshift.io/rescan=** 옵션을 사용하여 검사에 주석을 답니다.

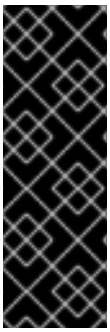
```
$ oc annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

재검사는 **rhcos-moderate** 프로파일에 대해 **4**개의 추가 **mc** 를 생성합니다.

```
$ oc get mc
```

출력 예

```
75-worker-scan-chronyd-or-ntpd-specify-remote-server
75-worker-scan-configure-usbguard-auditbackend
75-worker-scan-service-usbguard-enabled
75-worker-scan-usbguard-allow-hid-and-hub
```



중요

검사 설정 **default-auto-apply** 레이블이 적용되면 수정 사항이 자동으로 적용되고 오래된 수정 사항이 자동으로 업데이트됩니다. 종속 항목 또는 오래된 수정 사항으로 인해 적용되지 않은 업데이트 적용이 있는 경우 다시 검사하면 업데이트가 적용되고 재부팅이 트리거될 수 있습니다. **MachineConfig** 오브젝트를 사용하는 업데이트 적용만 재부팅을 트리거합니다. 적용할 업데이트 또는 종속 항목이 없는 경우 재부팅이 수행되지 않습니다.

5.10.4. 결과에 대한 사용자 정의 스토리지 크기 설정

ComplianceCheckResult와 같은 사용자 정의 리소스는 검사한 모든 노드에서 집계한 한 번의 점검 결과를 나타내지만 스캐너에서 생성한 원시 결과를 검토하는 것이 유용할 수 있습니다. 원시 결과는 **ARF** 형식으로 생성되며 크기가 클 수 있습니다(노드당 수십 메가바이트). 따라서 **etcd** 키-값 저장소에서 지원하는 **Kubernetes** 리소스에 저장하는 것은 비현실적입니다. 대신 검사할 때마다 기본 크기가 **1GB**인 영구 볼륨(**PV**)이 생성됩니다. 환경에 따라 적절하게 **PV** 크기를 늘릴 수 있습니다. 크기를 늘리려면 **ScanSetting** 및 **ComplianceScan** 리소스 모두에 노출되는 **rawResultStorage.size** 특성을 사용하면 됩니다.

관련 매개변수는 **rawResultStorage.rotation**으로, 이전 검사가 되풀이되기 전에 **PV**에 유지되는 검사 수를 조절합니다. 기본값은 **3**이며 되풀이 정책을 **0**으로 설정하면 되풀이가 비활성화됩니다. 기본 되풀이 정책과 원시 **ARF** 검사 보고서당 **100MB**의 추정치가 지정되면 환경에 적합한 **PV** 크기를 계산할 수 있습니다.

5.10.4.1. 사용자 정의 결과 스토리지 값 사용

OpenShift Container Platform은 다양한 퍼블릭 클라우드 또는 베어 메탈에 배포할 수 있으므로 **Compliance Operator**에서는 사용 가능한 스토리지 구성을 결정할 수 없습니다. 기본적으로 **Compliance Operator**는 클러스터의 기본 스토리지 클래스를 사용하여 결과를 저장하는 **PV**를 생성하지만 사용자 정의 스토리지 클래스는 **rawResultStorage.StorageClassName** 특성을 사용하여 구성할 수 있습니다.

중요

클러스터에서 기본 스토리지 클래스를 지정하지 않는 경우 이 특성을 설정해야 합니다.

표준 스토리지 클래스를 사용하고 마지막 결과 **10**개를 유지하는 **10GB** 크기의 영구 볼륨을 만들도록 **ScanSetting** 사용자 정의 리소스를 구성합니다.

예제 ScanSetting CR

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  storageClassName: standard
  rotation: 10
  size: 10Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
```

5.10.5. 제품군 검사에서 생성된 수정 사항 적용

ComplianceSuite 오브젝트에서 autoApplyRemediations 부울 매개 변수를 사용할 수 있지만, 대신 compliance.openshift.io/apply-remediations로 오브젝트에 주석을 달 수 있습니다. 이를 통해 Operator는 생성된 모든 수정 사항을 적용할 수 있습니다.

프로세스

- 다음을 실행하여 compliance.openshift.io/apply-remediations 주석을 적용합니다.

```
$ oc annotate compliancesuites/<suite-_name> compliance.openshift.io/apply-remediations=
```

5.10.6. 수정 사항 자동 업데이트

경우에 따라 최신 콘텐츠가 있는 검사에서 OUTDATED로 업데이트 적용을 표시할 수 있습니다. 관리자는 compliance.openshift.io/remove-outdated 주석을 적용하여 새 업데이트를 적용하고 오래된 항목을 제거할 수 있습니다.

프로세스

- compliance.openshift.io/remove-outdated 주석을 적용합니다.

```
$ oc annotate compliancesuites/<suite_name> compliance.openshift.io/remove-outdated=
```

또는 ScanSetting 또는 ComplianceSuite 오브젝트에 autoUpdateRemediations 플래그를 설정하여 수정 사항을 자동으로 업데이트합니다.

5.11. COMPLIANCE OPERATOR 문제 해결

이 섹션에서는 Compliance Operator 문제 해결 방법에 대해 설명합니다. 이 정보는 문제를 진단하거나 버그 보고서에 정보를 제공하는 데 유용할 수 있습니다. 몇 가지 일반적인 정보:

- Compliance Operator는 중요한 일이 발생할 때 Kubernetes 이벤트를 생성합니다. 다음 명령을 사용하여 클러스터의 모든 이벤트를 볼 수 있습니다.

```
$ oc get events -n openshift-compliance
```

또는 다음 명령을 사용하여 검사와 같은 오브젝트 이벤트를 볼 수 있습니다.

```
$ oc describe compliancescan/<scan_name>
```

- Compliance Operator는 대략 API 오브젝트당 하나씩 여러 개의 컨트롤러로 구성됩니다. 문제가 있는 API 오브젝트에 해당하는 컨트롤러만 필터링하는 것이 유용할 수 있습니다. ComplianceRemediation을 적용할 수 없는 경우 remediationctrl 컨트롤러의 메시지를 확인하십시오. jq를 사용하여 구문 분석하면 단일 컨트롤러의 메시지를 필터링할 수 있습니다.

```
$ oc logs compliance-operator-775d7bddbd-gj58f | jq -c 'select(.logger ==
"profilebundlectrl")'
```

- 타임스탬프는 UTC의 UNIX epoch 이후의 초로 기록됩니다. 사람이 읽을 수 있는 날짜로 변환하려면 date -d @timestamp --utc를 사용하십시오. 예를 들면 다음과 같습니다.

```
$ date -d @1596184628.955853 --utc
```

- 많은 사용자 정의 리소스 중에서도 가장 중요한 ComplianceSuite 및 ScanSetting에서는 debug 옵션을 설정할 수 있습니다. 이 옵션을 활성화하면 OpenSCAP 스캐너 Pod 및 기타 도우미 Pod의 상세 수준이 높아집니다.

- 단일 규칙이 예기치 않게 통과 또는 실패하는 경우 해당 규칙만 사용하여 단일 스캔 또는 모음을 실행하고 해당 **ComplianceCheckResult** 오브젝트에서 규칙 **ID**를 찾은 후 이를 **Scan CR**에서 **rule** 특성 값으로 사용하는 것이 도움이 될 수 있습니다. 그러면 **debug** 옵션이 활성화된 상태에서 스캐너 **Pod**의 **scanner** 컨테이너 로그에 원시 **OpenSCAP** 로그가 표시됩니다.

5.11.1. 검사 구조

다음 섹션에서는 **Compliance Operator** 검사의 구성 요소 및 단계를 간략하게 설명합니다.

5.11.1.1. 규정 준수 소스

규정 준수 콘텐츠는 **ProfileBundle** 오브젝트에서 생성되는 **Profile** 오브젝트에 저장됩니다. **Compliance Operator**는 클러스터와 클러스터 노드에 대해 각각 하나의 **ProfileBundle** 오브젝트를 생성합니다.

```
$ oc get profilebundle.compliance
```

```
$ oc get profile.compliance
```

ProfileBundle 오브젝트는 **Bundle**이라는 이름으로 레이블이 지정된 배포에서 처리합니다. **Bundle** 문제를 해결하려면 배포를 찾은 후 해당 배포에서 **Pod** 로그를 보면 됩니다.

```
$ oc logs -lprofile-bundle=ocp4 -c profileparser
```

```
$ oc get deployments,pods -lprofile-bundle=ocp4
```

```
$ oc logs pods/<pod-name>
```

```
$ oc describe pod/<pod-name> -c profileparser
```

5.11.1.2. ScanSetting 및 ScanSettingBinding 오브젝트 라이프사이클 및 디버깅

유효한 준수 콘텐츠 소스를 사용하면 높은 수준의 **ScanSetting** 및 **ScanSettingBinding** 오브젝트를 사용하여 **ComplianceSuite** 및 **ComplianceScan** 오브젝트를 생성할 수 있습니다.

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: my-companys-constraints
debug: true
# For each role, a separate scan will be created pointing
# to a node-role specified in roles
```

```
  roles:
    - worker
  ---
  apiVersion: compliance.openshift.io/v1alpha1
  kind: ScanSettingBinding
  metadata:
    name: my-companys-compliance-requirements
  profiles:
    # Node checks
    - name: rhcos4-e8
      kind: Profile
      apiGroup: compliance.openshift.io/v1alpha1
    # Cluster checks
    - name: ocp4-e8
      kind: Profile
      apiGroup: compliance.openshift.io/v1alpha1
  settingsRef:
    name: my-companys-constraints
    kind: ScanSetting
    apiGroup: compliance.openshift.io/v1alpha1
```

ScanSetting 및 ScanSettingBinding 오브젝트는 모두 logger=scansettingbindingctrl 태그가 지정된 동일한 컨트롤러에서 처리합니다. 이러한 오브젝트에는 상태가 없습니다. 모든 문제는 이벤트 형식으로 전달됩니다.

```
Events:
  Type    Reason       Age   From                 Message
  ----    ------       ----  ----                 -------
  Normal  SuiteCreated  9m52s  scansettingbindingctrl  ComplianceSuite openshift-
compliance/my-companys-compliance-requirements created
```

이제 ComplianceSuite 오브젝트가 생성되었습니다. flow는 새로 생성된 ComplianceSuite를 계속 조정합니다.

5.11.1.3. ComplianceSuite 사용자 정의 리소스 라이프사이클 및 디버깅

ComplianceSuite CR은 ComplianceScan 관련 래퍼입니다. ComplianceSuite CR은 logger=suitectrl 태그가 지정된 컨트롤러에서 처리합니다. 이 컨트롤러는 모음의 검사 생성을 처리하고 개별 검사 상태를 단일 모음 상태로 조정 및 집계합니다. 모음이 주기적으로 실행되도록 설정된 경우에는 초기 실행이 완료된 후 suitectrl에서도 CronJob CR 생성을 처리합니다.

```
$ oc get cronjobs
```

출력 예

```
NAME                            SCHEDULE    SUSPEND   ACTIVE   LAST SCHEDULE   AGE
<cron_name>                       0 1 * * *   False     0        <none>          151m
```

가장 중요한 문제의 경우 이벤트가 생성됩니다. **oc describe compliancesuites/<name>**으로 문제를 확인하십시오. **Suite** 오브젝트에는 이 모음에 속한 **Scan** 오브젝트에서 **Status** 하위 리소스를 업데이트할 때 업데이트되는 **Status** 하위 리소스도 있습니다. 예상되는 모든 검사가 생성되면 제어가 검사 컨트롤러로 전달됩니다.

5.11.1.4. ComplianceScan 사용자 정의 리소스 라이프사이클 및 디버깅

ComplianceScan CR은 **scanctrl** 컨트롤러에 의해 처리됩니다. 여기에서 실제 검사가 발생하고 검사 결과가 생성됩니다. 각 검사는 여러 단계를 거칩니다.

5.11.1.4.1. 보류 단계

이 단계에서는 검사의 정확성을 확인합니다. 스토리지 크기와 같은 일부 매개변수가 잘못된 경우 검사가 '오류와 함께 완료' 결과로 전환되고, 그러지 않으면 시작 단계가 진행됩니다.

5.11.1.4.2. 시작 단계

이 단계에서는 스캐너 **Pod**에 대한 환경이나 스캐너 **Pod**를 평가할 스크립트를 직접 포함하는 여러 구성 맵이 있습니다. 구성 맵을 나열합니다.

```
$ oc get cm -lcompliance.openshift.io/scan-name=rhcos4-e8-
worker,complianceoperator.openshift.io/scan-script=
```

이러한 구성 맵은 스캐너 **Pod**에서 사용합니다. 스캐너 동작을 수정하거나 스캐너 디버그 수준을 변경하거나 원시 결과를 출력해야 하는 경우 구성 맵을 수정하는 것이 좋습니다. 이후 원시 **ARF** 결과를 저장하기 위해 검사별 영구 볼륨 클레임이 생성됩니다.

```
$ oc get pvc -lcompliance.openshift.io/scan-name=<scan_name>
```

PVC는 검사별 **ResultServer** 배포로 마운트됩니다. **ResultServer**는 개별 스캐너 **Pod**에서 전체 **ARF** 결과를 업로드하는 간단한 **HTTP** 서버입니다. 각 서버는 다른 노드에서 실행될 수 있습니다. 전체 **ARF** 결과는 매우 클 수 있으며 동시에 여러 노드에서 마운트할 수 있는 볼륨을 생성할 수 있다고 가정해서는 안 됩니다. 검사가 완료되면 **ResultServer** 배포가 축소됩니다. 원시 결과가 있는 **PVC**는 다른 사용자 정의 **Pod**에서 마운트할 수 있으며 결과를 가져오거나 검사할 수 있습니다. 스캐너 **Pod**와 **ResultServer** 간 트래픽은 상호 **TLS** 프로토콜로 보호됩니다.

마지막으로 이 단계에서는 스캐너 **Pod**가 시작되는데, **Platform** 검사 인스턴스용 스캐너 **Pod 1**개와 **node** 검사 인스턴스에 일치하는 노드당 스캐너 **Pod 1**개입니다. 노드별 **Pod**는 노드 이름으로 레이블이 지정됩니다. 각 **Pod**는 항상 **ComplianceScan**이라는 이름으로 레이블이 지정됩니다.

```
$ oc get pods -lcompliance.openshift.io/scan-name=rhcos4-e8-worker,workload=scanner --
show-labels
```

출력 예

```
NAME                                                READY   STATUS    RESTARTS   AGE   LABELS
rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod   0/2     Completed   0
39m   compliance.openshift.io/scan-name=rhcos4-e8-worker,targetNode=ip-10-0-169-90.eu-
north-1.compute.internal,workload=scanner
At this point, the scan proceeds to the Running phase.
```

5.11.1.4.3. 실행 단계

실행 단계는 스캐너 **Pod**가 종료될 때까지 대기합니다. 다음은 실행 단계에서 사용되는 용어 및 프로세스입니다.

- **Init** 컨테이너: **content-container** 라는 하나의 **init** 컨테이너가 있습니다. **contentImage** 컨테이너를 실행하고 이 **Pod**의 다른 컨테이너와 공유하는 **/content** 디렉터리에 **contentFile**을 복사하는 단일 명령을 실행합니다.

- **scanner:** 이 컨테이너는 스캔을 실행합니다. 노드 검사의 경우 컨테이너는 노드 파일 시스템을 **/host**로 마운트하고 **init** 컨테이너에서 제공하는 콘텐츠를 마운트합니다. 컨테이너는 또한 시작 단계에서 생성된 **entrypoint ConfigMap**을 마운트하고 실행합니다. 진입점 **ConfigMap**의 기본 스크립트는 **OpenSCAP**을 실행하고 **Pod** 컨테이너 간 공유하는 **/results** 디렉터리에 결과 파일을 저장합니다. 이 **Pod**의 로그를 보고 **OpenSCAP** 스캐너에서 점검한 사항을 확인할 수 있습니다. 더 자세한 출력 내용은 **debug** 플래그를 사용하여 볼 수 있습니다.

- **logcollector: logcollector** 컨테이너는 **scanner** 컨테이너가 종료될 때까지 기다립니다. 그런 다음 전체 **ARF** 결과를 **ResultServer**에 업로드하고 **XCCDF** 결과를 검사 결과 및 **OpenSCAP** 결과 코드와 함께 **ConfigMap**으로 별도로 업로드합니다. 이러한 결과 구성 맵은 검사 이름 **(compliance.openshift.io/scan-name=<scan_name>)**으로 레이블이 지정됩니다.

```
$ oc describe cm/rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
```

출력 예

출력 예

```
Name:         rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
Namespace:    openshift-compliance
Labels:       compliance.openshift.io/scan-name-scan=rhcos4-e8-worker
              complianceoperator.openshift.io/scan-result=
Annotations:  compliance-remediations/processed:
              compliance.openshift.io/scan-error-msg:
              compliance.openshift.io/scan-result: NON-COMPLIANT
              OpenSCAP-scan-result/node: ip-10-0-169-90.eu-north-1.compute.internal

Data
====
exit-code:
----
2
results:
----
<?xml version="1.0" encoding="UTF-8"?>
...
```

Platform 검사를 위한 스캐너 Pod는 다음을 제외하고 비슷합니다.

- api-resource-collector라는 하나의 추가 init 컨테이너가 있습니다. 이 컨테이너는 content-container init 컨테이너에서 제공하는 OpenSCAP 콘텐츠를 읽고, 해당 콘텐츠에서 검사해야 하는 API 리소스를 파악한 후 scanner 컨테이너에서 이러한 API 리소스를 읽는 공유 디렉터리에 저장합니다.
- scanner 컨테이너는 호스트 파일 시스템을 마운트할 필요가 없습니다.

스캐너 Pod가 완료되면 검사가 집계 단계로 이동합니다.

5.11.1.4.4. 집계 단계

검사 컨트롤러는 집계 단계에서 집계기 Pod라는 또 다른 Pod를 생성합니다. 그 목적은 결과 ConfigMap 오브젝트를 가져와서 결과를 읽고 각 점검 결과에 해당하는 Kubernetes 오브젝트를 만드는 것입니다. 점검 실패를 자동으로 수정할 수 있는 경우 ComplianceRemediation 오브젝트가 생성됩니다. 또한 집계기 Pod는 사람이 읽을 수 있는 점검 및 수정용 메타데이터를 제공하기 위해 init 컨테이너를 사용하여 OpenSCAP 콘텐츠도 마운트합니다.

집계기 Pod에서 구성 맵을 처리하면 구성 맵에 compliance-remediations/processed라는 레이블이 지정됩니다. 이 단계의 결과는 ComplianceCheckResult 오브젝트

```
$ oc get compliancecheckresults -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

출력 예

```
NAME                                          STATUS   SEVERITY
rhcos4-e8-worker-accounts-no-uid-except-zero              PASS     high
rhcos4-e8-worker-audit-rules-dac-modification-chmod       FAIL     medium
```

및 ComplianceRemediation 오브젝트입니다.

```
$ oc get complianceremediations -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

출력 예

```
NAME                                          STATE
rhcos4-e8-worker-audit-rules-dac-modification-chmod       NotApplied
rhcos4-e8-worker-audit-rules-dac-modification-chown       NotApplied
rhcos4-e8-worker-audit-rules-execution-chcon              NotApplied
rhcos4-e8-worker-audit-rules-execution-restorecon         NotApplied
rhcos4-e8-worker-audit-rules-execution-semanage           NotApplied
rhcos4-e8-worker-audit-rules-execution-setfiles           NotApplied
```

이러한 CR이 생성되면 집계기 Pod가 종료되고 검사가 완료 단계로 전환됩니다.

5.11.1.4.5. 완료 단계

최종 검사 단계에서는 필요한 경우 검사 리소스가 정리되고 ResultServer 배포가 축소되거나(검사가 1회인 경우) 삭제됩니다(검사가 계속되는 경우). 삭제하는 경우 다음 검사 인스턴스에서 배포를 다시 생성합니다.

또한 주석을 달아 완료 단계에서 검사 재실행을 트리거할 수도 있습니다.

```
$ oc annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

autoApplyRemediations: true를 사용하여 수정을 자동 적용하도록 설정하지 않으면 검사가 완료 단계에 도달한 후 자체적으로 수행되는 작업이 없습니다. **OpenShift Container Platform** 관리자가 수정 사항을 검토하고 필요에 따라 적용합니다. 수정을 자동 적용하도록 설정하면 완료 단계에서 **ComplianceSuite** 컨트롤러에 설정이 전달되고, 이 컨트롤러에서 검사가 매핑되는 머신 구성 풀을 일시 중지한 후 모든 수정을 한 번에 적용합니다. 수정은 **ComplianceRemediation** 컨트롤러에서 대신 적용합니다.

5.11.1.5. ComplianceRemediation 컨트롤러 라이프사이클 및 디버깅

예제 검사에서 몇 가지 결과가 보고되었습니다. **apply** 특성을 **true**로 전환하여 수정 중 하나를 활성화할 수 있습니다.

```
$ oc patch complianceremediations/rhcos4-e8-worker-audit-rules-dac-modification-chmod --
patch '{"spec":{"apply":true}}' --type=merge
```

ComplianceRemediation 컨트롤러(**logger=remediationctrl**)는 수정된 오브젝트를 조정합니다. 조정으로 인해 조정된 수정 오브젝트의 상태가 변경될 뿐만 아니라 적용된 모든 수정이 포함되는 렌더링된 모음별 **MachineConfig** 오브젝트도 변경됩니다.

MachineConfig 오브젝트는 항상 **75-**로 시작하며 검사 및 모음 이름을 따서 이름이 지정됩니다.

```
$ oc get mc | grep 75-
```

출력 예

```
75-rhcos4-e8-worker-my-companys-compliance-requirements                                2.2.0
2m46s
```

현재 **mc**가 구성되어 있는 수정이 머신 구성의 주석에 나열됩니다.

```
$ oc describe mc/75-rhcos4-e8-worker-my-companys-compliance-requirements
```

출력 예

```
Name:         75-rhcos4-e8-worker-my-companys-compliance-requirements
Labels:       machineconfiguration.openshift.io/role=worker
Annotations:  remediation/rhcos4-e8-worker-audit-rules-dac-modification-chmod:
```

ComplianceRemediation 컨트롤러 알고리즘은 다음과 같이 작동합니다.

- 현재 적용된 모든 수정은 초기 수정 세트로 해석됩니다.
- 조정된 수정을 적용해야 하는 경우 이 세트에 추가됩니다.
- **MachineConfig** 오브젝트는 해당 세트에서 렌더링되고 세트의 수정 이름으로 주석이 달립니다. 세트가 비어 있는 경우(마지막 수정이 적용되지 않음) 렌더링된 **MachineConfig** 오브젝트가 제거됩니다.
- 렌더링된 머신 구성이 클러스터에 이미 적용된 구성과 다른 경우에만 적용된 **MC**가 업데이트되거나 생성 또는 삭제됩니다.
- **MachineConfig** 오브젝트를 생성하거나 수정하면 **machineconfiguration.openshift.io/role** 레이블과 일치하는 노드의 재부팅이 트리거됩니다. 자세한 내용은 **Machine Config Operator** 설명서를 참조하십시오.

필요한 경우 렌더링된 머신 구성을 업데이트하고 조정된 수정 오브젝트 상태를 업데이트하면 수정 루프가 종료됩니다. 예제의 경우 수정을 적용하면 재부팅이 트리거됩니다. 재부팅 후 검사에 주석을 달아 다시 실행합니다.

```
$ oc annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

검사가 실행되고 완료됩니다. 전달할 수정을 확인합니다.

```
$ oc get compliancecheckresults/rhcos4-e8-worker-audit-rules-dac-modification-chmod
```

출력 예

```
NAME                                           STATUS   SEVERITY
rhcos4-e8-worker-audit-rules-dac-modification-chmod   PASS     medium
```

5.11.1.6. 유용한 레이블

Compliance Operator에서 생성하는 각 Pod는 특별히 해당 Pod가 속하는 검사와 수행하는 작업을 사용하여 레이블이 지정됩니다. 검사 식별자는 compliance.openshift.io/scan-name으로 레이블이 지정됩니다. 워크로드 식별자는 workload로 레이블이 지정됩니다.

Compliance Operator는 다음 워크로드를 예약합니다.

- scanner: 규정 준수 스캔을 수행합니다.
- resultserver: 규정 준수 스캔의 원시 결과를 저장합니다.
- aggregator: 결과를 집계하고, 불일치를 감지하고, 결과 오브젝트를 출력합니다(검사 결과 및 수정).
- suitererunner: 은 다시 실행할 슈트에 태그를 지정합니다(일정이 설정된 경우).
- profileparser: 데이터 스트림을 구문 분석하고 적절한 프로필, 규칙 및 변수를 생성합니다.

특정 워크로드에 디버깅 및 로그가 필요한 경우 다음을 실행합니다.

```
$ oc logs -l workload=<workload_name> -c <container_name>
```

5.11.2. 지원 요청

이 문서에 설명된 절차 또는 일반적으로 OpenShift Container Platform에 문제가 발생하는 경우 Red

Hat 고객 포털에 액세스하십시오. 고객 포털에서 다음을 수행할 수 있습니다.

- Red Hat 제품과 관련된 기사 및 솔루션에 대한 Red Hat 지식베이스를 검색하거나 살펴볼 수 있습니다.
- Red Hat 지원에 대한 지원 케이스 제출할 수 있습니다.
- 다른 제품 설명서에 액세스 가능합니다.

클러스터 문제를 식별하려면 OpenShift Cluster Manager 에서 Insights를 사용할 수 있습니다. Insights는 문제에 대한 세부 정보 및 문제 해결 방법에 대한 정보를 제공합니다.

이 문서를 개선하기 위한 제안이 있거나 오류를 발견한 경우 가장 관련 있는 문서 구성 요소에 대한 Jira 문제를 제출하십시오. 섹션 이름 및 OpenShift Container Platform 버전과 같은 구체적인 정보를 제공합니다.

5.12. COMPLIANCE OPERATOR 설치 제거

OpenShift Container Platform 웹 콘솔을 사용하여 클러스터에서 OpenShift Compliance Operator를 제거할 수 있습니다.

5.12.1. OpenShift Container Platform에서 OpenShift Compliance Operator 설치 제거

Compliance Operator를 제거하려면 먼저 Compliance Operator CRD(사용자 정의 리소스 정의)를 삭제해야 합니다. CRD가 제거되면 openshift-compliance 프로젝트를 삭제하여 Operator 및 해당 네임스페이스를 제거할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- OpenShift Compliance Operator가 설치되어 있어야 합니다.

프로세스

OpenShift Container Platform 웹 콘솔을 사용하여 Compliance Operator를 제거하려면 다음을 수행합니다.

1. **Compliance Operator에서 설치한 CRD를 제거합니다.**

   a. **관리 → 사용자 정의 리소스 정의 페이지로 전환합니다.**

   b. **이름 필드에서 compliance.openshift.io를 검색합니다.**

   c. **다음 각 CRD 옆에 있는 옵션 메뉴**

   를 클릭하고 사용자 정의 리소스 정의 삭제를 선택합니다.

   - **ComplianceCheckResult**
   - **ComplianceRemediation**
   - **ComplianceScan**
   - **ComplianceSuite**
   - **ProfileBundle**
   - **Profile**
   - **Rule**
   - **ScanSettingBinding**

- ScanSetting
- TailoredProfile
- Variable

2. OpenShift Compliance 프로젝트를 제거합니다.

a. 홈 → 프로젝트 페이지로 전환합니다.

b. openshift-compliance 프로젝트 옆에 있는 옵션 메뉴  를 클릭하고 프로젝트 삭제를 선택합니다.

c. 대화 상자에 openshift-compliance를 입력하여 삭제를 확인하고 삭제를 클릭합니다.

5.13. 사용자 정의 리소스 정의 이해

OpenShift Container Platform의 Compliance Operator는 여러 CRD(Custom Resource Definitions)를 제공하여 규정 준수 검사를 수행합니다. 규정 준수 검사를 실행하려면 ComplianceAsCode 커뮤니티 프로젝트에서 파생된 사전 정의된 보안 정책을 활용합니다. Compliance Operator는 이러한 보안 정책을 CRD로 변환하여 규정 준수 검사를 실행하고 발견된 문제에 대한 수정을 가져올 수 있습니다.

5.13.1. CRD 워크플로

CRD는 다음 워크플로우를 제공하여 규정 준수 검사를 완료합니다.

1. 규정 준수 검사 요구 사항 정의
2. 규정 준수 검사 설정 구성

3. 규정 준수 검사 설정을 통한 프로세스 준수 요구 사항

4. 규정 준수 검사 모니터링

5. 규정 준수 검사 결과 확인

5.13.2. 컴플라이언스 스캔 요구 사항 정의

기본적으로 **Compliance Operator CRD**에는 **ProfileBundle** 및 **Profile** 오브젝트가 포함되며 규정 준수 검사 요구 사항에 대한 규칙을 정의하고 설정할 수 있습니다. **TailoredProfile** 오브젝트를 사용하여 기본 프로필을 사용자 지정할 수도 있습니다.

5.13.2.1. ProfileBundle 오브젝트

Compliance Operator를 설치하면 **ready-to-run ProfileBundle** 오브젝트가 포함됩니다. **Compliance Operator**는 **ProfileBundle** 오브젝트를 구문 분석하고 번들의 각 프로필에 대한 **Profile** 오브젝트를 생성합니다. 또한 **Profile** 개체에서 사용하는 규칙 및 변수 개체를 구문 분석합니다.

ProfileBundle 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
  name: <profile bundle name>
  namespace: openshift-compliance
spec:
  contentFile: ssg-ocp4-ds.xml 1
  contentImage: quay.io/complianceascode/ocp4:latest 2
status:
  dataStreamStatus: VALID 3
```

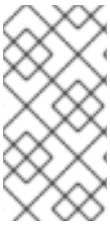
1 프로필 파일이 있는 루트 디렉터리(/)의 경로를 지정합니다.

2

프로필 파일을 캡슐화하는 컨테이너 이미지를 지정합니다.

3

Compliance Operator에서 콘텐츠 파일을 구문 분석할 수 있는지 여부를 나타냅니다.



참고

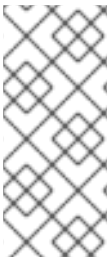
contentFile 이 실패하면 발생한 오류에 대한 세부 정보를 제공하는 **errorMessage** 속성이 표시됩니다.

문제 해결

잘못된 이미지에서 알려진 콘텐츠 이미지로 롤백하면 **ProfileBundle** 오브젝트에 응답하지 않고 **PENDING** 상태가 표시됩니다. 해결 방법으로 이전 이미지와 다른 이미지로 이동할 수 있습니다. 또는 **ProfileBundle** 오브젝트를 삭제하고 다시 만들어 작동 상태로 돌아갈 수 있습니다.

5.13.2.2. profile 오브젝트

Profile 오브젝트는 특정 규정 준수 표준에 대해 평가할 수 있는 규칙과 변수를 정의합니다. **XCCDF** 식별자 및 노드 또는 플랫폼 유형에 대한 프로필 검사와 같은 **OpenSCAP** 프로필에 대한 세부 정보를 구문 분석했습니다. **Profile** 오브젝트를 직접 사용하거나 **Tailor Profile** 오브젝트를 사용하여 추가로 사용자 지정할 수 있습니다.



참고

Profile 오브젝트는 단일 **ProfileBundle** 오브젝트에서 파생되므로 수동으로 생성하거나 수정할 수 없습니다. 일반적으로 단일 **ProfileBundle** 오브젝트에는 여러 **Profile** 오브젝트가 포함될 수 있습니다.

Profile 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
description: <description of the profile>
id: xccdf_org.ssgproject.content_profile_moderate 1
kind: Profile
metadata:
  annotations:
    compliance.openshift.io/product: <product name>
```

```
    compliance.openshift.io/product-type: Node 2
  creationTimestamp: "YYYY-MM-DDTMM:HH:SSZ"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: <profile bundle name>
  name: rhcos4-moderate
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: <profile bundle name>
    uid: <uid string>
  resourceVersion: "<version number>"
  selfLink: /apis/compliance.openshift.io/v1alpha1/namespaces/openshift-
compliance/profiles/rhcos4-moderate
  uid: <uid string>
rules: 3
- rhcos4-account-disable-post-pw-expiration
- rhcos4-accounts-no-uid-except-zero
- rhcos4-audit-rules-dac-modification-chmod
- rhcos4-audit-rules-dac-modification-chown
title: <title of the profile>
```

1

프로필의 **XCCDF** 이름을 지정합니다. **ComplianceScan** 오브젝트를 검사의 **profile** 속성 값으로 정의할 때 이 식별자를 사용합니다.

2

노드 또는 **Platform** 을 지정합니다. 노드 프로필은 클러스터 노드 및 플랫폼 프로필을 스캔하여 **Kubernetes** 플랫폼을 스캔합니다.

3

프로필의 규칙 목록을 지정합니다. 각 규칙은 하나의 검사에 해당합니다.

5.13.2.3. rule 오브젝트

프로필을 생성하는 **Rule** 오브젝트도 오브젝트로 노출됩니다. **Rule** 오브젝트를 사용하여 규정 준수 점검 요구 사항을 정의하고 수정 방법을 지정합니다.

Rule 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
checkType: Platform (1)
description: <description of the rule>
id: xccdf_org.ssgproject.content_rule_configure_network_policies_namespaces (2)
instructions: <manual instructions for the scan>
kind: Rule
metadata:
  annotations:
    compliance.openshift.io/rule: configure-network-policies-namespaces
    control.compliance.openshift.io/CIS-OCP: 5.3.2
    control.compliance.openshift.io/NERC-CIP: CIP-003-3 R4;CIP-003-3 R4.2;CIP-003-3
      R5;CIP-003-3 R6;CIP-004-3 R2.2.4;CIP-004-3 R3;CIP-007-3 R2;CIP-007-3 R2.1;CIP-007-3
      R2.2;CIP-007-3 R2.3;CIP-007-3 R5.1;CIP-007-3 R6.1
    control.compliance.openshift.io/NIST-800-53: AC-4;AC-4(21);CA-3(5);CM-6;CM-6(1);CM-
7;CM-7(1);SC-7;SC-7(3);SC-7(5);SC-7(8);SC-7(12);SC-7(13);SC-7(18)
  labels:
    compliance.openshift.io/profile-bundle: ocp4
  name: ocp4-configure-network-policies-namespaces
  namespace: openshift-compliance
rationale: <description of why this rule is checked>
severity: high (3)
title: <summary of the rule>
```

(1) 이 규칙이 실행되는 확인 유형을 지정합니다. 노드 프로필은 클러스터 노드 및 플랫폼 프로필을 스캔하여 **Kubernetes** 플랫폼을 검사합니다. 빈 값은 자동 검사가 없음을 나타냅니다.

(2) 데이터 스트림에서 직접 구문 분석되는 규칙의 **XCCDF** 이름을 지정합니다.

(3) 실패한 규칙의 심각도를 지정합니다.

참고

Rule 오브젝트는 연결된 **ProfileBundle** 오브젝트를 쉽게 식별하기 위한 적절한 레이블을 가져옵니다. **ProfileBundle** 은 이 오브젝트의 **OwnerReferences** 에도 지정됩니다.

5.13.2.4. TailoredProfile object

TailoredProfile 오브젝트를 사용하여 조직 요구 사항에 따라 기본 **Profile** 오브젝트를 수정합니다. 규칙을 활성화 또는 비활성화하고, 변수 값을 설정하고, 사용자 지정에 대한 타당성을 제공할 수 있습니다. 검증 후 **TailoredProfile** 오브젝트는 **ComplianceScan** 오브젝트에서 참조할 수 있는 **ConfigMap** 을 생성합니다.

작은 정보

ScanSettingBinding 오브젝트에서 참조하는 **TailoredProfile** 오브젝트를 사용할 수 있습니다. **ScanSettingBinding**에 대한 자세한 내용은 **ScanSettingBinding** 오브젝트를 참조하십시오.

TailoredProfile 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: rhcos4-with-usb
spec:
  extends: rhcos4-moderate (1)
  title: <title of the tailored profile>
  disableRules:
    - name: <name of a rule object to be disabled>
      rationale: <description of why this rule is checked>
status:
  id: xccdf_compliance.openshift.io_profile_rhcos4-with-usb (2)
  outputRef:
    name: rhcos4-with-usb-tp (3)
    namespace: openshift-compliance
  state: READY (4)
```

(1)

이는 선택 사항입니다. **Tailored Profile** 이 빌드된 **Profile** 오브젝트의 이름입니다. 값을 설정하지 않으면 **enableRules** 목록에서 새 프로필이 생성됩니다.

(2)

맞춤형 프로필의 **XCCDF** 이름을 지정합니다.

(3)

ComplianceScan 의 **tailoringConfigMap.name** 속성 값으로 사용할 수 있는 **ConfigMap** 이름을 지정합니다.

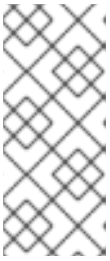
4

READY,PENDING, FAILURE 와 같은 오브젝트의 상태를 표시합니다. 오브젝트의 상태가 ERROR 이면 status.errorMessage 는 실패 이유를 제공합니다.

TailoredProfile 오브젝트를 사용하면 TailoredProfile 구문을 사용하여 새 Profile 오브젝트를 생성할 수 있습니다. 새 프로필 을 생성하려면 다음 구성 매개변수를 설정합니다.

- 적절한 제목
- 확장 값이 비어 있어야 합니다.
- TailoredProfile 오브젝트에서 검사 유형 주석:

```
compliance.openshift.io/product-type: <scan type>
```



참고

product-type 주석을 설정하지 않은 경우 Compliance Operator는 기본적으로 Platform 검사 유형으로 설정됩니다. TailoredProfile 오브젝트의 이름에 -node 접미사를 추가하면 노드 검사 유형이 생성됩니다.

5.13.3. 규정 준수 검사 설정 구성

컴플라이언스 검사의 요구 사항을 정의한 후에는 검사 유형, 검사 발생 및 검사 위치를 지정하여 구성할 수 있습니다. 이를 위해 Compliance Operator는 ScanSetting 오브젝트를 제공합니다.

5.13.3.1. ScanSetting 오브젝트

ScanSetting 오브젝트를 사용하여 운영 정책을 정의하고 재사용하여 스캔을 실행합니다. 기본적으로 Compliance Operator는 다음 ScanSetting 오브젝트를 생성합니다.

- 기본값 - 1Gi 영구 볼륨(PV)을 사용하여 마스터 노드와 작업자 노드 모두에서 1 AM에 대한 검사를 실행하고 마지막 세 가지 결과를 유지합니다. 수정은 자동으로 적용되거나 업데이트되지 않습니다.

- **default-auto-apply - 1Gi Persistent Volume(PV)**을 사용하여 컨트롤 플레인과 작업자 노드에서 매일 **1AM** 스캔을 실행하고 마지막 세 가지 결과를 유지합니다. **autoApplyRemediations** 및 **autoUpdateRemediations** 가 모두 **true**로 설정됩니다.

예제 **ScanSetting** 오브젝트

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: <name of the scan>
autoApplyRemediations: false 1
autoUpdateRemediations: false 2
schedule: "0 1 * * *" 3
rawResultStorage:
  size: "2Gi" 4
  rotation: 10 5
roles: 6
  - worker
  - master
```

1 자동 수정을 활성화하려면 **true** 로 설정합니다. 자동 수정을 비활성화하려면 **false** 로 설정합니다.

2 콘텐츠 업데이트에 대한 자동 수정을 활성화하려면 **true** 로 설정합니다. 콘텐츠 업데이트에 대한 자동 수정을 비활성화하려면 **false** 로 설정합니다.

3 **cron** 형식으로 검사를 실행해야 하는 빈도를 지정합니다.

4 원시 결과를 저장할 검사에 대해 생성해야 하는 스토리지 크기를 지정합니다. 기본값은 **1Gi**입니다.

5 원시 결과가 저장될 검사 양을 지정합니다. 기본값은 **3**입니다. 이전 결과가 교체되면 관리자는 순환이 발생하기 전에 다른 위치에 결과를 저장해야 합니다.

참고

순환 정책을 비활성화하려면 값을 0 으로 설정합니다.

6

노드 유형에 대한 검사를 예약하려면 node-role.kubernetes.io 레이블 값을 지정합니다. 이 값은 MachineConfigPool 의 이름과 일치해야 합니다.

5.13.4. 규정 준수 검사 설정을 사용하여 규정 준수 검사 요구 사항 처리

규정 준수 검사 요구 사항을 정의하고 검사를 실행하도록 설정을 구성한 경우 Compliance Operator는 ScanSettingBinding 오브젝트를 사용하여 처리합니다.

5.13.4.1. ScanSettingBinding object

ScanSettingBinding 오브젝트를 사용하여 Profile 또는 TailoredProfile 오브젝트에 대한 참조로 규정 준수 요구 사항을 지정합니다. 그런 다음 검사에 대한 운영 제한 조건을 제공하는 ScanSetting 오브젝트에 연결됩니다. 그런 다음 Compliance Operator는 ScanSetting 및 ScanSettingBinding 오브젝트를 기반으로 ComplianceSuite 오브젝트를 생성합니다.

예제 ScanSettingBinding 오브젝트

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: <name of the scan>
profiles: 1
  # Node checks
  - name: rhcos4-with-usb
    kind: TailoredProfile
    apiGroup: compliance.openshift.io/v1alpha1
  # Cluster checks
  - name: ocp4-moderate
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef: 2
  name: my-companys-constraints
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

1

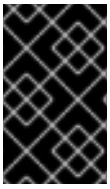
환경을 스캔할 **Profile** 또는 **TailoredProfile** 오브젝트의 세부 정보를 지정합니다.

2

일정 및 스토리지 크기와 같은 운영 제약 조건을 지정합니다.

ScanSetting 및 **ScanSettingBinding** 오브젝트를 생성하면 규정 준수 제품군이 생성됩니다. 규정 준수 제품군 목록을 가져오려면 다음 명령을 실행합니다.

```
$ oc get compliancesuites
```



중요

ScanSettingBinding 을 삭제하면 규정 준수 제품군도 삭제됩니다.

5.13.5. 규정 준수 검사 추적

규정 준수 제품군 생성 후 **ComplianceSuite** 오브젝트를 사용하여 배포된 검사의 상태를 모니터링할 수 있습니다.

5.13.5.1. ComplianceSuite 오브젝트

ComplianceSuite 오브젝트는 검사 상태를 추적하는 데 도움이 됩니다. 여기에는 검사 및 전체 결과를 생성하는 원시 설정이 포함되어 있습니다.

노드 유형 검사의 경우 문제에 대한 수정이 포함되어 있으므로 검사를 **MachineConfigPool** 에 매핑해야 합니다. 레이블을 지정하는 경우 풀에 직접 적용되는지 확인합니다.

ComplianceSuite 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
```

```
  name: <name of the scan>
spec:
  autoApplyRemediations: false (1)
  schedule: "0 1 * * *" (2)
  scans: (3)
    - name: workers-scan
      scanType: Node
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: quay.io/complianceascode/ocp4:latest
      rule: "xccdf_org.ssgproject.content_rule_no_netrc_files"
      nodeSelector:
        node-role.kubernetes.io/worker: ""
status:
  Phase: DONE (4)
  Result: NON-COMPLIANT (5)
  scanStatuses:
  - name: workers-scan
    phase: DONE
    result: NON-COMPLIANT
```

(1) 자동 수정을 활성화하려면 **true** 로 설정합니다. 자동 수정을 비활성화하려면 **false** 로 설정합니다.

(2) **cron** 형식으로 검사를 실행해야 하는 빈도를 지정합니다.

(3) 클러스터에서 실행할 검사 사양 목록을 지정합니다.

(4) 검사 진행 상황을 나타냅니다.

(5) 제품군의 전체 **verdict**를 나타냅니다.

백그라운드의 제품군은 **scans** 매개변수를 기반으로 **ComplianceScan** 오브젝트를 생성합니다. 프로그래밍 방식으로 **ComplianceSuites** 이벤트를 가져올 수 있습니다. 제품군에 대한 이벤트를 가져오려면

다음 명령을 실행합니다.

```
$ oc get events --field-selector involvedObject.kind=ComplianceSuite,involvedObject.name=
<name of the suite>
```

중요

XCCDF 속성이 포함되어 있으므로 ComplianceSuite 를 수동으로 정의할 때 오류를 생성할 수 있습니다.

5.13.5.2. 고급 ComplianceScan 오브젝트

Compliance Operator에는 디버깅 또는 기존 툴링과의 통합을 위한 고급 사용자용 옵션이 포함되어 있습니다. ComplianceScan 오브젝트를 직접 생성하지 않는 것이 좋지만 ComplianceSuite 오브젝트를 사용하여 대신 관리할 수 있습니다.

Advanced ComplianceScan 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceScan
metadata:
  name: <name of the scan>
spec:
  scanType: Node 1
  profile: xccdf_org.ssgproject.content_profile_moderate 2
  content: ssg-ocp4-ds.xml
  contentImage: quay.io/complianceascode/ocp4:latest 3
  rule: "xccdf_org.ssgproject.content_rule_no_netrc_files" 4
  nodeSelector: 5
    node-role.kubernetes.io/worker: ""
status:
  phase: DONE 6
  result: NON-COMPLIANT 7
```

1

노드 또는 Platform 을 지정합니다. 노드 프로필은 클러스터 노드 및 플랫폼 프로필을 스캔하여 Kubernetes 플랫폼을 스캔합니다.

2

3

프로필 파일을 캡슐화하는 컨테이너 이미지를 지정합니다.

4

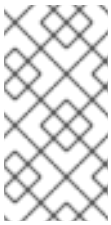
이는 선택 사항입니다. 단일 규칙을 실행하려면 검사를 지정합니다. 이 규칙은 **XCCDF ID**로 식별되어야 하며 지정된 프로필에 속해야 합니다.

참고

rule 매개변수를 건너뛰면 지정된 프로필의 사용 가능한 모든 규칙에 대해 검사가 실행됩니다.

5

OpenShift Container Platform에 있고 수정을 생성하려는 경우 **nodeSelector** 레이블은 **MachineConfigPool** 레이블과 일치해야 합니다.



참고

nodeSelector 매개변수를 지정하지 않거나 **MachineConfig** 레이블과 일치하는 경우 검사가 계속 실행되지만 수정을 생성하지는 않습니다.

6

검사의 현재 단계를 나타냅니다.

7

검사의 유효성 검사를 나타냅니다.

중요

ComplianceSuite 오브젝트를 삭제하면 연결된 모든 검사가 삭제됩니다.

검사가 완료되면 **ComplianceCheckResult** 오브젝트의 **Custom Resources**로 결과가 생성됩니다. 그러나 원시 결과는 **ARF** 형식으로 제공됩니다. 이러한 결과는 검사 이름과 연결된 **PVC**(영구 볼륨 클레

임)가 있는 PV(영구 볼륨)에 저장됩니다. 프로그래밍 방식으로 ComplianceScans 이벤트를 가져올 수 있습니다. 제품군에 대한 이벤트를 생성하려면 다음 명령을 실행합니다.

```
oc get events --field-selector involvedObject.kind=ComplianceScan,involvedObject.name=
<name of the suite>
```

5.13.6. 규정 준수 결과 보기

규정 준수 제품군이 DONE 단계에 도달하면 검사 결과 및 가능한 수정을 확인할 수 있습니다.

5.13.6.1. ComplianceCheckResult 오브젝트

특정 프로필과 함께 검사를 실행하면 프로필의 여러 규칙이 확인됩니다. 이러한 각 규칙에 대해 특정 규칙에 대해 클러스터 상태를 제공하는 ComplianceCheckResult 오브젝트가 생성됩니다.

ComplianceCheckResult 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceCheckResult
metadata:
  labels:
    compliance.openshift.io/check-severity: medium
    compliance.openshift.io/check-status: FAIL
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
  name: workers-scan-no-direct-root-logins
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceScan
    name: workers-scan
description: <description of scan check>
instructions: <manual instructions for the scan>
id: xccdf_org.ssgproject.content_rule_no_direct_root_logins
severity: medium (1)
status: FAIL (2)
```

(1)

2

검사 결과에 대해 설명합니다. 가능한 값은 다음과 같습니다.

- **PassS:** 검사가 성공적으로 수행되었습니다.
- **FAIL:** 검사가 실패했습니다.
- **INFO:** 검사가 성공적으로 수행되었으며 오류로 간주될 수 있을 만큼 심각하지 않은 것이 발견되었습니다.
- **MANUAL:** 검사가 자동으로 상태를 평가할 수 없으며 수동 검사가 필요합니다.
- **INCONSISTENT:** 다른 노드에서 다른 결과를 보고합니다.
- **ERROR:** 확인 실행이 성공적으로 수행되지만 완료할 수 없습니다.
- **NotappLICABLE:** 적용할 수 없으므로 검사가 실행되지 않았습니다.

제품군에서 모든 검사 결과를 가져오려면 다음 명령을 실행합니다.

```
oc get compliancecheckresults -l compliance.openshift.io/suite=<suit name>
```

5.13.6.2. ComplianceRemediation 오브젝트

특정 검사의 경우 데이터 스트림을 지정된 수정 사항을 사용할 수 있습니다. 그러나 Kubernetes 수정 사항을 사용할 수 있는 경우 Compliance Operator에서 ComplianceRemediation 오브젝트를 생성합니다.

ComplianceRemediation 오브젝트의 예

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  labels:
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
    machineconfiguration.openshift.io/role: worker
  name: workers-scan-disable-users-coredumps
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
    name: workers-scan-disable-users-coredumps
    uid: <UID>
spec:
  apply: false (1)
  object:
    current: (2)
      apiVersion: machineconfiguration.openshift.io/v1
      kind: MachineConfig
      spec:
        config:
          ignition:
            version: 2.2.0
          storage:
            files:
            - contents:
                source: data:,%2A%20%20%20%20%20hard%20%20%20core%20%20%20%200
              filesystem: root
              mode: 420
              path: /etc/security/limits.d/75-disable_users_coredumps.conf
    outdated: {} (3)
```

(1) **true** 는 수정 사항이 적용되었음을 나타냅니다. **false** 는 수정 사항이 적용되지 않았음을 나타냅니다.

(2) 수정 정의 포함

(3) 이전에 이전 버전의 콘텐츠에서 구문 분석된 수정을 나타냅니다. **Compliance Operator**는 여전히 오래된 오브젝트를 유지하여 관리자에게 적용하기 전에 새 수정을 검토할 수 있는 기회를 제공합니다.

제품군에서 모든 수정을 가져오려면 다음 명령을 실행합니다.

```
oc get complianceremediations -l compliance.openshift.io/suite=<suite name>
```

자동으로 복구할 수 있는 실패한 모든 검사를 나열하려면 다음 명령을 실행합니다.

```
oc get compliancecheckresults -l 'compliance.openshift.io/check-status in
(FAIL),compliance.openshift.io/automated-remediation'
```

수동으로 수정할 수 있는 실패한 모든 검사를 나열하려면 다음 명령을 실행합니다.

```
oc get compliancecheckresults -l 'compliance.openshift.io/check-status in
(FAIL),!compliance.openshift.io/automated-remediation'
```

6장. FILE INTEGRITY OPERATOR

6.1. FILE INTEGRITY OPERATOR 릴리스 노트

OpenShift Container Platform용 File Integrity Operator는 RHCOS 노드에서 파일 무결성 검사를 지속적으로 실행합니다.

이 릴리스 노트는 OpenShift Container Platform의 File Integrity Operator 개발을 추적합니다.

File Integrity Operator 개요는 File Integrity Operator 이해를 참조하십시오.

6.1.1. OpenShift File Integrity Operator 0.1.30

OpenShift File Integrity Operator 0.1.30에서 다음 권고를 사용할 수 있습니다.

- RHBA-2022:5538 OpenShift File Integrity Operator 버그 수정 및 기능 업데이트

6.1.1.1. 버그 수정

- 이전에는 File Integrity Operator에서 발행한 경고가 네임스페이스를 설정하지 않아 경고가 시작된 위치를 파악하기 어려웠습니다. 이제 Operator에서 적절한 네임스페이스를 설정하여 경고에 대한 이해를 높입니다. (BZ#2101393)

6.1.2. OpenShift File Integrity Operator 0.1.24

OpenShift File Integrity Operator 0.1.24에 다음 권고를 사용할 수 있습니다.

- RHBA-2022:1331 OpenShift File Integrity Operator 버그 수정

6.1.2.1. 새로운 기능 및 개선 사항

- 이제 config.maxBackups 속성을 사용하여 FileIntegrity 사용자 정의 리소스(CR)에 저장된 최대 백업 수를 구성할 수 있습니다. 이 속성은 노드에 유지할 수 있도록 AIDE 데이터베이스 수와 re-init 프로세스에서 남은 로그 백업을 지정합니다. 구성된 번호 이외의 이전 백업이 자동으로 정리됩니다. 기본값은 5개의 백업으로 설정됩니다.

6.1.2.2. 버그 수정

- 이전 버전에서는 0.1.21 이전 버전에서 0.1.22로 Operator를 업그레이드하면 다시 초기화 기능이 실패할 수 있었습니다. 이로 인해 Operator가 configMap 리소스 라벨을 업데이트하지 못했습니다. 이제 최신 버전으로 업그레이드하면 리소스 레이블이 수정됩니다. (BZ#2049206)

- 이전에는 기본 configMap 스크립트 콘텐츠를 적용할 때 잘못된 데이터 키가 비교되었습니다. 이로 인해 Operator 업그레이드 후 aide-reinit 스크립트가 올바르게 업데이트되지 않아 re-init 프로세스가 실패했습니다. 이제daemonSets 가 완료될 때까지 실행되고 AIDE 데이터베이스 다시 시작 프로세스가 성공적으로 실행됩니다. (BZ#2072058)

6.1.3. OpenShift File Integrity Operator 0.1.22

다음 권고는 OpenShift File Integrity Operator 0.1.22에 제공됩니다.

- RHBA-2022:0142 OpenShift File Integrity Operator 버그 수정

6.1.3.1. 버그 수정

- 이전에는 File Integrity Operator가 설치된 시스템에서 /etc/kubernetes/aide.reinit 파일로 인해 OpenShift Container Platform 업데이트가 중단될 수 있었습니다. 이는 /etc/kubernetes/aide.reinit 파일이 존재하지만 나중에 ostree 검증 전에 제거된 경우 발생했습니다. 이번 업데이트를 통해 /etc/kubernetes/aide.reinit 가 OpenShift Container Platform 업데이트와 충돌하지 않도록 /run 디렉터리로 이동합니다. (BZ#2033311)

6.1.4. OpenShift File Integrity Operator 0.1.21

다음 권고는 OpenShift File Integrity Operator 0.1.21에 사용할 수 있습니다.

- RHBA-2021:4631 OpenShift File Integrity Operator 버그 수정 및 개선 사항 업데이트

6.1.4.1. 새로운 기능 및 개선 사항

- FileIntegrity 스캔 결과 및 처리 지표와 관련된 지표는 웹 콘솔의 모니터링 대시보드에 표시됩니다. 결과는 file_integrity_operator_ 접두사로 레이블이 지정됩니다.

- 노드에 1초 이상의 무결성 오류가 있는 경우 Operator 네임스페이스에 제공된 기본 PrometheusRule 경고에 경고가 표시됩니다.

- 다음 동적 **Machine Config Operator** 및 **Cluster Version Operator** 관련 파일 경로는 노드 업데이트 중에 오탐을 방지하기 위해 기본 **AIDE** 정책에서 제외됩니다.
  - **/etc/machine-config-daemon/currentconfig**
  - **/etc/pki/ca-trust/extracted/java/cacerts**
  - **/etc/cvo/updatepayloads**
  - **/root/.kube**
- **AIDE** 데몬 프로세스에는 **v0.1.16**보다 안정성이 개선되어 있으며 **AIDE** 데이터베이스를 초기화할 때 발생할 수 있는 오류에 대한 복원력이 향상됩니다.

6.1.4.2. 버그 수정

- 이전에는 **Operator**가 자동으로 업그레이드할 때 오래된 데몬 세트가 제거되지 않았습니다. 이번 릴리스에서는 자동 업그레이드 중에 오래된 데몬 세트가 제거됩니다.

6.1.5. 추가 리소스

- File Integrity Operator 이해

6.2. FILE INTEGRITY OPERATOR 설치

6.2.1. 웹 콘솔을 사용하여 File Integrity Operator 설치

사전 요구 사항

- **admin** 권한이 있어야 합니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 Operator → OperatorHub로 이동합니다.

2. File Integrity Operator 를 검색한 다음 설치를 클릭합니다.

3. 기본 설치 모드 및 네임스페이스를 계속 선택하여 Operator가 openshift-file-integrity 네임스페이스에 설치되도록 합니다.

4. 설치를 클릭합니다.

검증

설치에 성공했는지 확인하려면 다음을 수행하십시오.

1. Operator → 설치된 Operator 페이지로 이동합니다.

2. Operator가 openshift-file-integrity 네임스페이스에 설치되어 있고 해당 상태는 Succeeded인지 확인합니다.

Operator가 성공적으로 설치되지 않은 경우 다음을 수행하십시오.

1. Operator → 설치된 Operator 페이지로 이동하여 Status 열에 오류 또는 실패가 있는지 점검합니다.

2. 워크로드 → Pod 페이지로 전환하고 openshift-file-integrity 프로젝트에서 문제를 보고하는 Pod의 로그를 확인합니다.

6.2.2. CLI를 사용하여 File Integrity Operator 설치

사전 요구 사항

- admin 권한이 있어야 합니다.

프로세스

1. 다음을 실행하여 **Namespace** 오브젝트 **YAML** 파일을 생성합니다.

```
$ oc create -f <file-name>.yaml
```

출력 예

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-file-integrity
```

2. **OperatorGroup** 오브젝트 **YAML** 파일을 생성합니다.

```
$ oc create -f <file-name>.yaml
```

출력 예

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  targetNamespaces:
  - openshift-file-integrity
```

3. **Subscription** 오브젝트 **YAML** 파일을 생성합니다.

```
$ oc create -f <file-name>.yaml
```

출력 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  channel: "release-0.1"
  installPlanApproval: Automatic
  name: file-integrity-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

검증

1. CSV 파일을 검사하여 설치에 성공했는지 확인합니다.

```
$ oc get csv -n openshift-file-integrity
```

2. File Integrity Operator가 실행 중인지 확인합니다.

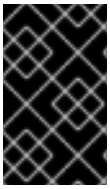
```
$ oc get deploy -n openshift-file-integrity
```

6.2.3. 추가 리소스

- File Integrity Operator는 제한된 네트워크 환경에서 지원됩니다. 자세한 내용은 제한된 네트워크에서 Operator Lifecycle Manager 사용을 참조하십시오.

6.3. FILE INTEGRITY OPERATOR 이해

File Integrity Operator는 클러스터 노드에서 파일 무결성 검사를 지속적으로 실행하는 OpenShift Container Platform Operator입니다. 이 Operator는 각 노드에서 권한 있는 AIDE(고급 침입 탐지 환경) 컨테이너를 초기화 및 실행하는 데몬 세트를 배포하여 데몬 세트 Pod 초기 실행 중 수정된 파일의 로그를 상태 오브젝트에 제공합니다.



중요

현재는 **RHCOS(Red Hat Enterprise Linux CoreOS)** 노드만 지원됩니다.

6.3.1. FileIntegrity 사용자 정의 리소스 생성

FileIntegrity 사용자 정의 리소스**(CR)** 인스턴스는 하나 이상의 노드에 대한 연속적인 파일 무결성 검사 세트를 나타냅니다.

각 **FileIntegrity CR**은 **FileIntegrity CR** 사양과 일치하는 노드에서 **AIDE**를 실행하는 데몬 세트에서 지원합니다.

프로세스

1. 작업자 노드에서 검사를 활성화하려면 **worker-fileintegrity.yaml** 이라는 다음 예제 **FileIntegrity CR**을 생성합니다.

FileIntegrity CR의 예

```
apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: worker-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  config: {}
```

2. **openshift-file-integrity** 네임스페이스에 **YAML** 파일을 적용합니다.

```
$ oc apply -f worker-fileintegrity.yaml -n openshift-file-integrity
```

검증

- 다음 명령을 실행하여 **FileIntegrity** 오브젝트가 성공적으로 생성되었는지 확인합니다.

```
$ oc get fileintegrities -n openshift-file-integrity
```

출력 예

```
NAME                 AGE
worker-fileintegrity   14s
```

6.3.2. FileIntegrity 사용자 정의 리소스 상태 확인

FileIntegrity 사용자 정의 리소스(CR)는 .status.phase 하위 리소스를 통해 해당 상태를 보고합니다.

프로세스

- FileIntegrity CR 상태를 쿼리하려면 다음을 실행합니다.

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status.phase }"
```

출력 예

```
Active
```

6.3.3. FileIntegrity 사용자 정의 리소스 단계

- Pending - 사용자 정의 리소스(CR)가 생성된 후 단계입니다.
- Active - 백업 데몬 세트가 설정되어 실행되는 단계입니다.
- Initializing - AIDE 데이터베이스가 다시 초기화되는 단계입니다.

6.3.4. FileIntegrityNodeStatuses 오브젝트 이해

FileIntegrity CR의 검사 결과는 FileIntegrityNodeStatuses라는 다른 오브젝트에 보고됩니다.

```
$ oc get fileintegritynodestatuses
```

출력 예

```
NAME                                                 AGE
worker-fileintegrity-ip-10-0-130-192.ec2.internal   101s
worker-fileintegrity-ip-10-0-147-133.ec2.internal   109s
worker-fileintegrity-ip-10-0-165-160.ec2.internal   102s
```

참고

FileIntegrityNodeStatus 오브젝트 결과를 사용할 수 있는 데 시간이 걸릴 수 있습니다.

노드당 하나의 결과 오브젝트가 있습니다. 각 FileIntegrityNodeStatus 오브젝트의 nodeName 특성은 검사 중인 노드에 해당합니다. 파일 무결성 검사의 상태는 results 배열에 표시되며 여기에는 검사 조건이 포함됩니다.

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

fileintegritynodestatus 오브젝트는 AIDE 실행의 최신 상태를 보고하고 status 필드에 Failed, Succeeded, Errored로 표시합니다.

```
$ oc get fileintegritynodestatuses -w
```

출력 예

```
NAME                                                 NODE                                                 STATUS
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal   ip-10-0-134-186.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal   ip-10-0-150-230.us-east-
```

```
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-169-137.us-east-2.compute.internal   ip-10-0-169-137.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal   ip-10-0-180-200.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal    ip-10-0-194-66.us-east-
2.compute.internal    Failed
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal   ip-10-0-222-188.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal   ip-10-0-134-186.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal   ip-10-0-222-188.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal    ip-10-0-194-66.us-east-
2.compute.internal    Failed
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal   ip-10-0-150-230.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal   ip-10-0-180-200.us-east-
2.compute.internal   Succeeded
```

6.3.5. FileIntegrityNodeStatus CR 상태 유형

이러한 조건은 해당 **FileIntegrityNodeStatus CR** 상태의 결과 배열에 보고됩니다.

- **Succeeded -** 무결성 검사를 통과했습니다. 데이터베이스가 마지막으로 초기화된 이후 **AIDE** 검사에 포함된 파일과 디렉터리가 수정되지 않았습니다.
- **Failed -** 무결성 검사에 실패했습니다. 데이터베이스가 마지막으로 초기화된 이후 **AIDE** 검사에 포함된 일부 파일 또는 디렉터리가 수정되었습니다.
- **Errored - AIDE** 스캐너에 내부 오류가 발생했습니다.

6.3.5.1. FileIntegrityNodeStatus CR 성공 상태의 예

성공 상태가 있는 조건의 출력 예

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:57Z"
```

```
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:46:03Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:48Z"
  }
]
```

이 경우 세 가지 검사가 모두 성공했으며 지금까지 다른 조건이 없습니다.

6.3.5.2. FileIntegrityNodeStatus CR 실패 상태의 예

실패 조건을 시뮬레이션하려면 **AIDE**가 추적하는 파일 중 하나를 수정하십시오. 예를 들어 작업자 노드 중 하나에서 **/etc/resolv.conf**를 수정합니다.

```
$ oc debug node/ip-10-0-130-192.ec2.internal
```

출력 예

```
Creating debug namespace/openshift-debug-node-ldfbj ...
Starting pod/ip-10-0-130-192ec2internal-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.0.130.192
If you don't see a command prompt, try pressing enter.
sh-4.2# echo "# integrity test" >> /host/etc/resolv.conf
sh-4.2# exit

Removing debug pod ...
Removing debug namespace/openshift-debug-node-ldfbj ...
```

잠시 후 해당 **FileIntegrityNodeStatus** 오브젝트의 결과 배열에 **Failed** 조건이 보고되었습니다. 이전의 **Succeeded** 조건이 유지되므로 검사가 실패한 시간을 정확히 찾을 수 있습니다.

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io/worker-fileintegrity-ip-10-0-130-
192.ec2.internal -ojsonpath='{.results}' | jq -r
```

또는 오브젝트 이름을 언급하지 않는 경우 다음을 실행합니다.

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

출력 예

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:54:14Z"
  },
  {
    "condition": "Failed",
    "filesChanged": 1,
    "lastProbeTime": "2020-09-15T12:57:20Z",
    "resultConfigMapName": "aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed",
    "resultConfigMapNamespace": "openshift-file-integrity"
  }
]
```

Failed 조건은 정확히 무엇이 실패하고 왜 실패했는지에 대한 자세한 정보를 제공하는 구성 맵을 가리킵니다.

```
$ oc describe cm aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
```

출력 예

```
Name:         aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
Namespace:    openshift-file-integrity
Labels:       file-integrity.openshift.io/node=ip-10-0-130-192.ec2.internal
              file-integrity.openshift.io/owner=worker-fileintegrity
              file-integrity.openshift.io/result-log=
Annotations:  file-integrity.openshift.io/files-added: 0
              file-integrity.openshift.io/files-changed: 1
              file-integrity.openshift.io/files-removed: 0
```

```
Data

integritylog:
------
AIDE 0.15.1 found differences between database and filesystem!!
Start timestamp: 2020-09-15 12:58:15

Summary:
  Total number of files:  31553
  Added files:              0
  Removed files:            0
  Changed files:            1


---------------------------------------------------
Changed files:
---------------------------------------------------

changed: /hostroot/etc/resolv.conf

---------------------------------------------------
Detailed information about changes:
---------------------------------------------------


File: /hostroot/etc/resolv.conf
 SHA512   : sTQYpB/AL7FeoGtu/1g7opv6C+KT1CBJ ,
qAeM+a8yTgHPnlHMaRlS+so61EN8VOpg

Events:  <none>
```

구성 맵 데이터 크기 제한으로 인해 **1MB** 이상의 **AIDE** 로그가 실패 구성 맵에 **base64**로 인코딩된 **gzip** 아카이브로 추가됩니다. 이 경우 위 명령의 출력을 **base64 --decode | gunzip** 으로 연결하려고 합니다. 압축 로그는 구성 맵에 **file-integrity.openshift.io/compressed** 주석 키가 있는 것으로 표시됩니다.

6.3.6. 이벤트 이해

FileIntegrity 및 **FileIntegrityNodeStatus** 오브젝트의 상태의 전환은 *이벤트*에서 기록됩니다. 이벤트 생성 시간은 **Active**로 **Initializing**과 같은 최신 전환을 반영하며 반드시 최신 검사 결과가 반영되는 것은 아닙니다. 그러나 최신 이벤트는 항상 최근 상태를 반영합니다.

```
$ oc get events --field-selector reason=FileIntegrityStatus
```

출력 예

```
LAST SEEN   TYPE     REASON              OBJECT                                MESSAGE
97s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Pending
67s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Initializing
37s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Active
```

노드 검사에 실패하면 **add/changed/removed** 및 **config map** 정보를 사용하여 이벤트가 생성됩니다.

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

출력 예

```
LAST SEEN   TYPE     REASON              OBJECT                                MESSAGE
114m        Normal   NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to
node ip-10-0-134-173.ec2.internal
114m        Normal   NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to
node ip-10-0-168-238.ec2.internal
114m        Normal   NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to
node ip-10-0-169-175.ec2.internal
114m        Normal   NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to
node ip-10-0-152-92.ec2.internal
114m        Normal   NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to
node ip-10-0-158-144.ec2.internal
114m        Normal   NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to
node ip-10-0-131-30.ec2.internal
87m         Warning  NodeIntegrityStatus   fileintegrity/example-fileintegrity   node ip-10-0-152-
92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-
fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

추가, 변경 또는 제거된 파일의 수를 변경하면 노드 상태가 전환되지 않은 경우에도 새 이벤트가 생성됩니다.

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

출력 예

```
LAST SEEN   TYPE     REASON              OBJECT                                MESSAGE
114m        Normal   NodeIntegrityStatus fileintegrity/example-fileintegrity   no changes to
node ip-10-0-134-173.ec2.internal
114m        Normal   NodeIntegrityStatus fileintegrity/example-fileintegrity   no changes to
node ip-10-0-168-238.ec2.internal
114m        Normal   NodeIntegrityStatus fileintegrity/example-fileintegrity   no changes to
node ip-10-0-169-175.ec2.internal
114m        Normal   NodeIntegrityStatus fileintegrity/example-fileintegrity   no changes to
node ip-10-0-152-92.ec2.internal
114m        Normal   NodeIntegrityStatus fileintegrity/example-fileintegrity   no changes to
node ip-10-0-158-144.ec2.internal
114m        Normal   NodeIntegrityStatus fileintegrity/example-fileintegrity   no changes to
node ip-10-0-131-30.ec2.internal
87m         Warning  NodeIntegrityStatus fileintegrity/example-fileintegrity   node ip-10-0-152-
92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-
fileintegrity-ip-10-0-152-92.ec2.internal-failed
40m         Warning  NodeIntegrityStatus fileintegrity/example-fileintegrity   node ip-10-0-152-
92.ec2.internal has changed! a:3,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-
fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

6.4. CUSTOM FILE INTEGRITY OPERATOR 구성

6.4.1. FileIntegrity 오브젝트 특성 보기

모든 Kubernetes 사용자 정의 리소스(CR)와 마찬가지로 **oc explain fileintegrity**를 실행하면 다음을 사용하여 개별 특성을 볼 수 있습니다.

```
$ oc explain fileintegrity.spec
```

```
$ oc explain fileintegrity.spec.config
```

6.4.2. 중요한 특성

표 6.1. 중요한 spec 및 spec.config 특성

속성	설명
spec.nodeSelector	해당 노드에 AIDE Pod를 예약하기 위해서는 노드의 레이블과 일치해야 하는 키-값 쌍에 대한 맵입니다. 일반적인 사용은 **node-role.kubernetes.io/worker: ""**로 모든 작업자 노드에서 AIDE를 예약하고, **node.openshift.io/os_id: "rhcos"**로 모든 RHCOS(Red Hat Enterprise Linux CoreOS) 노드에서 일정을 예약하는 단일 키-값 쌍만 설정하는 것입니다.

속성	설명
spec.debug	부울 특성입니다. **true**로 설정하면 AIDE 데몬 세트의 Pod에서 실행 중인 데몬에서 추가 정보를 출력합니다.
spec.tolerations	사용자 정의 테인트가 있는 노드에 예약할 허용 오차를 지정합니다. 지정하지 않으면 기본 허용 오차가 적용되어 컨트롤 플레인 노드 (마스터 노드라고도 함)에서 허용 오차가 실행될 수 있습니다.
spec.config.gracePeriod	AIDE 무결성 검사 사이에 일시 중지하는 시간(초)입니다. 노드에 대한 빈번한 AIDE 검사는 리소스 집약적일 수 있으므로 간격을 길게 지정하는 것이 유용할 수 있습니다. 기본값은 **900** 또는 15분입니다.
maxBackups	노드를 유지하기 위해 **re-init** 프로세스에서 남은 최대 AIDE 데이터베이스 및 로그 백업 수입니다. 이 수를 초과하는 이전 백업은 데몬에서 자동으로 정리합니다.
spec.config.name	사용자 지정 AIDE 구성이 포함된 configMap의 이름입니다. 생략하면 기본 구성이 생성됩니다.
spec.config.namespace	사용자 지정 AIDE 구성이 포함된 configMap의 네임스페이스입니다. 설정되지 않은 경우 FIO는 RHCOS 시스템에 적합한 기본 구성을 생성합니다.
spec.config.key	**이름** 및 **네임스페이스**로 지정된 구성 맵에 실제 AIDE 구성이 포함된 key입니다. 기본값은 **aide.conf** 입니다.

6.4.3. 기본 구성 검사

기본 **File Integrity Operator** 구성은 **FileIntegrity CR**과 동일한 이름으로 구성 맵에 저장됩니다.

프로세스

- 기본 구성을 검토하려면 다음을 실행합니다.

```
$ oc describe cm/worker-fileintegrity
```

6.4.4. 기본 File Integrity Operator 구성 이해

다음은 구성 맵의 **aide.conf** 키에서 발췌한 것입니다.

```
@@define DBDIR /hostroot/etc/kubernetes
@@define LOGDIR /hostroot/etc/kubernetes
database=file:@@{DBDIR}/aide.db.gz
database_out=file:@@{DBDIR}/aide.db.gz
gzip_dbout=yes
verbose=5
report_url=file:@@{LOGDIR}/aide.log
report_url=stdout
PERMS = p+u+g+acl+selinux+xattrs
CONTENT_EX = sha512+ftype+p+u+g+n+acl+selinux+xattrs

/hostroot/boot/     CONTENT_EX
/hostroot/root/\..* PERMS
/hostroot/root/   CONTENT_EX
```

FileIntegrity 인스턴스의 기본 구성은 다음 디렉터리의 파일에 대한 적용 범위를 제공합니다.

- **/root**
- **/boot**
- **/usr**
- **/etc**

다음 디렉터리에는 적용되지 않습니다.

- **/var**
- **/opt**
- **/etc/** 아래의 일부 **OpenShift** 관련 디렉터리는 제외됩니다.

6.4.5. 사용자 정의 AIDE 구성 제공

DBDIR, LOGDIR, database, database_out과 같은 **AIDE** 내부 동작을 구성하는 모든 항목을 **Operator**에서 덮어씁니다. **Operator**는 무결성 변경을 확인할 모든 경로 앞에 **/hostroot/** 접두사를 추가합

니다. 그러면 대부분 컨테이너화된 환경에 맞게 조정되지 않고 루트 디렉터리에서 더 쉽게 시작할 수 있는 기존 **AIDE** 구성이 재사용됩니다.

참고

/hostroot는 **AIDE**를 실행하는 **Pod**에서 호스트의 파일 시스템을 마운트하는 디렉터리입니다. 구성을 변경하면 데이터베이스가 다시 초기화됩니다.

6.4.6. 사용자 정의 File Integrity Operator 구성 정의

이 예제에서는 **worker-fileintegrity CR**에 제공된 기본 구성을 기반으로 컨트롤 플레인 노드 (마스터 노드라고도 함)에서 실행되는 스캐너의 사용자 정의 구성을 정의하는 데 중점을 둡니다. 이 워크플로는 데몬 세트로 실행되는 사용자 정의 소프트웨어를 배포하고 컨트롤 플레인 노드의 **/opt/mydaemon**에 데이터를 저장하려는 경우 유용할 수 있습니다.

프로세스

1. 기본 구성을 복사합니다.

2. 확인 또는 제외해야 하는 파일을 사용하여 기본 구성을 편집합니다.

3. 편집한 내용을 새 구성 맵에 저장합니다.

4. **spec.config**의 특성을 통해 **FileIntegrity** 오브젝트를 새 구성 맵으로 지정합니다.

5. 기본 구성을 추출합니다.

```
$ oc extract cm/worker-fileintegrity --keys=aide.conf
```

그러면 편집할 수 있는 **aide.conf**라는 파일이 생성됩니다. **Operator**에서 경로를 후처리하는 방법을 설명하기 위해 이 예제에서는 접두사 없이 제외 디렉터리를 추가합니다.

```
$ vim aide.conf
```

출력 예

```
/hostroot/etc/kubernetes/static-pod-resources
!/hostroot/etc/kubernetes/aide.*
!/hostroot/etc/kubernetes/manifests
!/hostroot/etc/docker/certs.d
!/hostroot/etc/selinux/targeted
!/hostroot/etc/openvswitch/conf.db
```

컨트롤 플레인 노드 관련 경로를 제외합니다.

```
!/opt/mydaemon/
```

다른 내용은 /etc에 저장합니다.

```
/hostroot/etc/ CONTENT_EX
```

6. 이 파일을 기반으로 구성 맵을 생성합니다.

```
$ oc create cm master-aide-conf --from-file=aide.conf
```

7. 구성 맵을 참조하는 FileIntegrity CR 매니페스트를 정의합니다.

```
apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: master-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector:
     node-role.kubernetes.io/master: ""
  config:
     name: master-aide-conf
     namespace: openshift-file-integrity
```

Operator는 제공된 구성 맵 파일을 처리하고 결과를 FileIntegrity 오브젝트와 동일한 이름의 구성 맵에 저장합니다.

```
$ oc describe cm/master-fileintegrity | grep /opt/mydaemon
```

출력 예

```
!/hostroot/opt/mydaemon
```

6.4.7. 사용자 정의 파일 무결성 구성 변경

파일 무결성 구성을 변경하려면 생성된 구성 맵을 변경하지 마십시오. 대신 **spec.name, namespace, key** 특성을 통해 **FileIntegrity** 오브젝트에 연결된 구성 맵을 변경하십시오.

6.5. 고급 CUSTOM FILE INTEGRITY OPERATOR 작업 수행

6.5.1. 데이터베이스 다시 초기화

File Integrity Operator에서 계획된 변경을 감지하면 데이터베이스를 다시 초기화해야 할 수 있습니다.

프로세스

- **file-integrity.openshift.io/re-init**로 **FileIntegrity** 사용자 정의 리소스(CR)에 주석을 답니다.

```
$ oc annotate fileintegrities/worker-fileintegrity file-integrity.openshift.io/re-init=
```

이전 데이터베이스 및 로그 파일이 백업되고 새 데이터베이스가 초기화됩니다. **oc debug**를 사용하여 생성된 **Pod**의 다음 출력에서 볼 수 있듯이 이전 데이터베이스 및 로그는 **/etc/kubernetes** 아래의 노드에 보관됩니다.

출력 예

```
 ls -IR /host/etc/kubernetes/aide.*
-rw-------. 1 root root 1839782 Sep 17 15:08 /host/etc/kubernetes/aide.db.gz
-rw-------. 1 root root 1839783 Sep 17 14:30 /host/etc/kubernetes/aide.db.gz.backup-
20200917T15_07_38
-rw-------. 1 root root   73728 Sep 17 15:07 /host/etc/kubernetes/aide.db.gz.backup-
20200917T15_07_55
-rw-r--r--. 1 root root       0 Sep 17 15:08 /host/etc/kubernetes/aide.log
-rw-------. 1 root root     613 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
```

```
20200917T15_07_38
-rw-r--r--. 1 root root      0 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_55
```

일부 레코드 영구성을 제공하기 위해 결과 구성 맵은 **FileIntegrity** 오브젝트에 속하지 않으므로 수동 정리가 필요합니다. 그 결과 **FileIntegrityNodeStatus** 오브젝트에 이전의 무결성 실패가 계속 표시됩니다.

6.5.2. 머신 구성 통합

OpenShift Container Platform 4에서 클러스터 노드 구성은 **MachineConfig** 오브젝트를 통해 제공됩니다. **MachineConfig** 오브젝트로 인해 파일 변경이 예상되며 이러한 변경으로 파일 무결성 검사가 실패하지 않아야 합니다. **MachineConfig** 오브젝트 업데이트로 인한 파일 변경을 억제하기 위해 **File Integrity Operator**에서 노드 오브젝트를 감시합니다. 노드가 업데이트될 때 업데이트 기간 동안 **AIDE** 검사가 일시 중단됩니다. 업데이트가 완료되면 데이터베이스가 다시 초기화되고 검사가 다시 시작됩니다.

이러한 일시 중지 및 재개 논리는 노드 오브젝트 주석에 반영되므로 **MachineConfig API**를 통한 업데이트에만 적용됩니다.

6.5.3. 데몬 세트 탐색

각 **FileIntegrity** 오브젝트는 여러 노드에 대한 검사를 나타냅니다. 검사 자체는 데몬 세트에서 관리하는 **Pod**에서 수행합니다.

FileIntegrity 오브젝트를 나타내는 데몬 세트를 찾으려면 다음을 실행하십시오.

```
$ oc -n openshift-file-integrity get ds/aide-worker-fileintegrity
```

해당 데몬 세트의 **Pod**를 나열하려면 다음을 실행합니다.

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

단일 **AIDE Pod**의 로그를 보려면 **Pod** 중 하나에서 **oc logs**를 호출합니다.

```
$ oc -n openshift-file-integrity logs pod/aide-worker-fileintegrity-mr8x6
```

출력 예

```
Starting the AIDE runner daemon
initializing AIDE db
initialization finished
running aide check
...
```

AIDE 데몬에서 생성한 구성 맵은 보관되지 않으며 File Integrity Operator에서 처리한 후 삭제됩니다. 그러나 실패 및 오류 시에는 이러한 구성 맵의 내용이 FileIntegrityNodeStatus 오브젝트가 가리키는 구성 맵에 복사됩니다.

6.6. FILE INTEGRITY OPERATOR 문제 해결

6.6.1. 일반 문제 해결

문제

File Integrity Operator의 일반적인 문제를 해결하려고 합니다.

해결

FileIntegrity 오브젝트에서 디버그 플래그를 활성화합니다. debug 플래그를 사용하면 DaemonSet Pod에서 실행되며 AIDE 검사를 실행하는 데몬의 상세 수준이 높아집니다.

6.6.2. AIDE 구성 확인

문제

AIDE 구성을 확인하려고 합니다.

해결

AIDE 구성은 FileIntegrity 오브젝트와 동일한 이름으로 구성 맵에 저장됩니다. 모든 AIDE 구성의 구성 맵에는 file-integrity.openshift.io/aide-conf 레이블이 지정됩니다.

6.6.3. FileIntegrity 오브젝트의 단계 확인

문제

FileIntegrity 오브젝트가 있는지 파악하고 현재 상태를 확인하려고 합니다.

해결

FileIntegrity 오브젝트의 현재 상태를 보려면 다음을 실행합니다.

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status }"
```

FileIntegrity 오브젝트와 백업 데몬 세트가 생성되면 상태가 Active로 전환되어야 합니다. 그러지 않으면 Operator Pod 로그를 확인하십시오.

6.6.4. 데몬 세트의 Pod가 예상 노드에서 실행 중인지 확인

문제

데몬 세트가 존재하고 해당 Pod가 실행되어야 하는 노드에서 실행되고 있는지 확인하려고 합니다.

해결

다음을 실행합니다.

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

참고

-owide 를 추가하면 Pod가 실행 중인 노드의 IP 주소가 포함됩니다.

데몬 Pod의 로그를 확인하려면 oc logs 를 실행합니다.

AIDE 명령의 반환 값을 확인하여 검사가 통과 또는 실패했는지 확인하십시오.

7장. 감사 로그 보기

OpenShift Container Platform 감사에서는 시스템의 개별 사용자, 관리자 또는 기타 구성 요소가 시스템에 영향을 준 활동 시퀀스를 설명하는 보안 관련 레코드 집합을 제공합니다.

7.1. API 감사 로그 정보

감사는 API 서버 수준에서 작동하며 서버로 들어오는 모든 요청을 기록합니다. 각 감사 로그에는 다음 정보가 포함됩니다.

표 7.1. 감사 로그 필드

필드	설명
level	이벤트가 생성된 감사 수준입니다.
auditID	각 요청에 생성되는 고유 감사 ID입니다.
stage	이 이벤트 인스턴스가 생성되었을 때의 요청 처리 단계입니다.
requestURI	클라이언트에서 서버로 보낸 요청 URI입니다.
verb	요청과 관련된 Kubernetes 동사입니다. 리소스가 아닌 요청의 경우 소문자 HTTP 메서드입니다.
user	인증된 사용자 정보입니다.
impersonatedUser	선택 사항입니다. 요청에서 다른 사용자를 가장하는 경우 가장된 사용자 정보입니다.
sourceIPs	선택 사항입니다. 요청이 발생한 소스 IP 및 중간 프록시입니다.
userAgent	선택 사항입니다. 클라이언트에서 보고한 사용자 에이전트 문자열입니다. 사용자 에이전트는 클라이언트에서 제공하며 신뢰할 수 없습니다.
objectRef	선택 사항입니다. 이 요청의 대상이 되는 오브젝트 참조입니다. 이는 **List** 유형의 요청 또는 리소스가 아닌 요청에는 적용되지 않습니다.
responseStatus	선택 사항입니다. **ResponseObject**가 **Status** 유형이 아닌 경우에도 채워지는 응답 상태입니다. 성공적인 응답을 위해 여기에는 코드만 포함됩니다. 상태 유형이 아닌 오류 응답의 경우 오류 메시지가 자동으로 채워집니다.

필드	설명
requestObject	선택 사항입니다. 요청의 API 오브젝트로, JSON 형식으로 되어 있습니다. **RequestObject**는 버전 변환, 기본값 설정, 승인 또는 병합 전에 요청에 있는 그대로(JSON으로 다시 인코딩될 수 있음) 기록됩니다. 외부 버전이 지정된 오브젝트 유형이며 그 자체로는 유효한 오브젝트가 아닐 수 있습니다. 리소스가 아닌 요청의 경우 생략되며 요청 수준 이상에서만 기록됩니다.
responseObject	선택 사항입니다. 응답에서 반환된 API 오브젝트로, JSON 형식으로 되어 있습니다. **ResponseObject**는 외부 유형으로 변환된 후 기록되고 JSON으로 직렬화됩니다. 리소스가 아닌 요청의 경우 생략되며 응답 수준에서만 기록됩니다.
requestReceivedTimestamp	요청이 API 서버에 도달한 시간입니다.
stageTimestamp	요청이 현재 감사 단계에 도달한 시간입니다.
annotations	선택 사항입니다. 인증, 권한 부여, 승인 플러그인을 포함하여 요청 서비스 체인에서 호출된 플러그인에 의해 설정될 수 있는 감사 이벤트와 함께 저장되는 구조화되지 않은 키 값 맵입니다. 이러한 주석은 감사 이벤트용이며 제출된 오브젝트의 **metadata.annotations**와 일치하지 않습니다. 키는 이름 충돌을 방지하기 위해 알림 구성 요소를 고유하게 식별해야 합니다(예: **podsecuritypolicy.admission.k8s.io/policy)**. 값은 짧아야 합니다. 주석은 메타데이터 수준에 포함됩니다.

Kubernetes API 서버의 출력 예:


```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"ad209ce1-fec7-4130-8192-c4cc63f1d8cd","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/openshift-kube-controller-manager/configmaps/cert-recovery-controller-lock?timeout=35s","verb":"update","user":{"username":"system:serviceaccount:openshift-kube-controller-manager:localhost-recovery-client","uid":"dd4997e3-d565-4e37-80f8-7fc122ccd785","groups":["system:serviceaccounts","system:serviceaccounts:openshift-kube-controller-manager","system:authenticated"]},"sourceIPs":["::1"],"userAgent":"cluster-kube-controller-manager-operator/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"configmaps","namespace":"openshift-kube-controller-manager","name":"cert-recovery-controller-lock","uid":"5c57190b-6993-425d-8101-8337e48c7548","apiVersion":"v1","resourceVersion":"574307"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2020-04-02T08:27:20.200962Z","stageTimestamp":"2020-04-02T08:27:20.206710Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:openshift:operator:kube-controller-manager-recovery\" of ClusterRole \"cluster-admin\" to ServiceAccount \"localhost-recovery-client/openshift-kube-controller-manager\""}}
```


7.2. 감사 로그 보기

각 컨트롤 플레인 노드 (마스터 노드라고도 함)에 대한 **OpenShift API** 서버, **Kubernetes API** 서버 및 **OpenShift OAuth API** 서버의 로그를 볼 수 있습니다.

프로세스

감사 로그를 보려면 다음을 수행합니다.

- **OpenShift API** 서버 서비스 로그 보기

a. 각 컨트롤 플레인 노드에 사용할 수 있는 **OpenShift API** 서버 로그를 나열합니다.

```
$ oc adm node-logs --role=master --path=openshift-apiserver/
```

출력 예

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T00-12-19.834.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T00-11-49.835.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T00-13-00.128.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. 노드 이름과 로그 이름을 지정하여 특정 **OpenShift API** 서버 로그를 확인합니다.

```
$ oc adm node-logs <node_name> --path=openshift-apiserver/<log_name>
```

예를 들면 다음과 같습니다.

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=openshift-
apiserver/audit-2021-03-09T00-12-19.834.log
```

출력 예

{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"381acf6d-5f30-4c7d-8175-c9c317ae5893","stage":"ResponseComplete","requestURI":"/metrics","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","uid":"825b60a0-3976-4861-a342-3b2b561e8f82","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"]},"sourceIPs":["10.129.2.6"],"userAgent":"Prometheus/2.23.0","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T18:02:04.086545Z","stageTimestamp":"2021-03-08T18:02:04.107102Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"prometheus-k8s\" of ClusterRole \"prometheus-k8s\" to ServiceAccount \"prometheus-k8s/openshift-monitoring\""}}

- **Kubernetes API** 서버 로그를 봅니다.

a. 각 컨트롤 플레인 노드에 사용할 수 있는 **Kubernetes API** 서버 로그를 나열합니다.

```
$ oc adm node-logs --role=master --path=kube-apiserver/
```

출력 예

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T14-07-27.129.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T19-24-22.620.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T18-37-07.511.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. 노드 이름과 로그 이름을 지정하여 특정 **Kubernetes API** 서버 로그를 확인합니다.

```
$ oc adm node-logs <node_name> --path=kube-apiserver/<log_name>
```

예를 들면 다음과 같습니다.

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=kube-
apiserver/audit-2021-03-09T14-07-27.129.log
```

출력 예

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"cfce8a
0b-b5f5-4365-8c9f-
79c1227d10f9","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/o
penshift-kube-scheduler/serviceaccounts/openshift-kube-scheduler-
sa","verb":"get","user":{"username":"system:serviceaccount:openshift-kube-
scheduler-operator:openshift-kube-scheduler-operator","uid":"2574b041-f3c8-
44e6-a057-baef7aa81516","groups":
["system:serviceaccounts","system:serviceaccounts:openshift-kube-scheduler-
operator","system:authenticated"]},"sourceIPs":
["10.128.0.8"],"userAgent":"cluster-kube-scheduler-operator/v0.0.0 (linux/amd64)
kubernetes/$Format","objectRef":
{"resource":"serviceaccounts","namespace":"openshift-kube-
scheduler","name":"openshift-kube-scheduler-
sa","apiVersion":"v1"},"responseStatus":{"metadata":
{},"code":200},"requestReceivedTimestamp":"2021-03-
08T18:06:42.512619Z","stageTimestamp":"2021-03-
08T18:06:42.516145Z","annotations":{"authentication.k8s.io/legacy-
token":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-
kube-scheduler-
operator","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"R
BAC: allowed by ClusterRoleBinding \"system:openshift:operator:cluster-kube-
scheduler-operator\" of ClusterRole \"cluster-admin\" to ServiceAccount
\"openshift-kube-scheduler-operator/openshift-kube-scheduler-operator\""}}
```

- **OpenShift OAuth API** 서버 로그 보기

a. 각 컨트롤 플레인 노드에서 사용할 수 있는 **OpenShift OAuth API** 서버 로그를 나열합니다.

```
$ oc adm node-logs --role=master --path=oauth-apiserver/
```

출력 예

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T13-06-26.128.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T18-23-21.619.log
```

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T17-36-06.510.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. 노드 이름과 로그 이름을 지정하여 특정 **OpenShift OAuth API** 서버 로그를 확인합니다.

```
$ oc adm node-logs <node_name> --path=oauth-apiserver/<log_name>
```

예를 들면 다음과 같습니다.

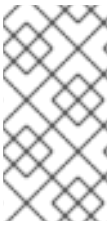
```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-
apiserver/audit-2021-03-09T13-06-26.128.log
```

출력 예

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"dd4c4
4e2-3ea1-4830-9ab7-
c91a5f1388d6","stage":"ResponseComplete","requestURI":"/apis/user.openshift.io
/v1/users/~","verb":"get","user":{"username":"system:serviceaccount:openshift-
monitoring:prometheus-k8s","groups":
["system:serviceaccounts","system:serviceaccounts:openshift-
monitoring","system:authenticated"]},"sourceIPs":
["10.0.32.4","10.128.0.1"],"userAgent":"dockerregistry/v0.0.0 (linux/amd64)
kubernetes/$Format","objectRef":
{"resource":"users","name":"~","apiGroup":"user.openshift.io","apiVersion":"v1"}
,"responseStatus":{"metadata":
{},"code":200},"requestReceivedTimestamp":"2021-03-
08T17:47:43.653187Z","stageTimestamp":"2021-03-
08T17:47:43.660187Z","annotations":
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC:
allowed by ClusterRoleBinding \"basic-users\" of ClusterRole \"basic-user\" to
Group \"system:authenticated\""}}
```

7.3. 감사 로그 필터링

jq 또는 다른 **JSON** 구문 분석 툴을 사용하여 **API** 서버 감사 로그를 필터링할 수 있습니다.



참고

설정된 감사 로그 정책에 의해 API 서버 감사 로그에 기록되는 정보의 양을 제어할 수 있습니다.

다음 절차에서는 jq를 사용하여 컨트롤 플레인 노드node-1.example.com에서 감사 로그를 필터링하는 예를 제공합니다. jq를 사용하는 방법에 대한 자세한 내요ㅇ은 jq 설명서를 참조하십시오.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- jq를 설치했습니다.

프로세스

- 사용자가 OpenShift API 서버 감사 로그를 필터링합니다.

```
$ oc adm node-logs node-1.example.com  \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.user.username == "myusername")'
```

- 사용자 에이전트가 OpenShift API 서버 감사 로그를 필터링합니다.

```
$ oc adm node-logs node-1.example.com  \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.userAgent == "cluster-version-operator/v0.0.0 (linux/amd64)
kubernetes/$Format")'
```

- 특정 API 버전 별로 Kubernetes API 서버 감사 로그를 필터링하고 사용자 에이전트만 출력합니다.

```
$ oc adm node-logs node-1.example.com  \
  --path=kube-apiserver/audit.log \
  | jq 'select(.requestURI | startswith("/apis/apiextensions.k8s.io/v1beta1")) |
.userAgent'
```

- 동사 제외하여 OpenShift OAuth API 서버 감사 로그를 필터링합니다.

```
$ oc adm node-logs node-1.example.com  \
  --path=oauth-apiserver/audit.log \
  | jq 'select(.verb != "get")'
```

7.4. 감사 로그 수집

must-gather 툴을 사용하여 클러스터를 디버깅하기 위한 감사 로그를 수집할 수 있으며 이를 통해 **Red Hat** 지원을 검토하거나 보낼 수 있습니다.

프로세스

1. **-- /usr/bin/gather_audit_logs** 플래그를 사용하여 **oc adm must-gather** 명령을 실행합니다.

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

2. 작업 디렉토리에서 생성된 **must-gather** 디렉토리에서 압축 파일을 만듭니다. 예를 들어 **Linux** 운영 체제를 사용하는 컴퓨터에서 다음 명령을 실행합니다.

```
$ tar cvaf must-gather.tar.gz must-gather.local.472290403699006248 1
```

1 **must-gather-local.4722403699006248** 을 실제 디렉터리 이름으로 교체합니다.

3. **Red Hat Customer Portal**에서 해당 지원 사례에 압축 파일을 첨부합니다.

7.5. 추가 리소스

- **must-gather** 툴
- **API** 감사 로그 이벤트 구조
- 감사 로그 정책 구성
- 타사 시스템에 로그 전달

8장. 감사 로그 정책 구성

이제 사용할 감사 로그 정책 프로필을 선택하여 API 서버 감사 로그에 기록되는 정보의 양을 제어할 수 있습니다.

8.1. 감사 로그 정책 프로필 정보

감사 로그 프로필은 OpenShift API 서버, Kubernetes API 서버, OAuth API 서버로 들어오는 요청을 기록하는 방법을 정의합니다.

OpenShift Container Platform에서는 다음과 같이 사전 정의된 감사 정책 프로필을 제공합니다.

Profile	설명
Default	읽기 및 쓰기 요청에 대한 메타데이터만 기록합니다. OAuth 액세스 토큰 생성(로그인) 요청을 제외하고 요청 본문은 기록하지 않습니다. 이는 기본 정책입니다.
WriteRequestBodies	모든 요청에 대한 메타데이터 기록 외에도 API 서버에 대한 모든 쓰기 요청(**create**, **update**, **patch**)의 요청 본문을 기록합니다. 이 프로필에는 **Default** 프로필보다 리소스 오버헤드가 많습니다.[1]
AllRequestBodies	모든 요청에 대한 메타데이터 기록 외에도 API 서버에 대한 모든 읽기 및 쓰기 요청(**get**, **list**, **create**, **update**, **patch**)의 요청 본문을 기록합니다. 이 프로필은 리소스 오버헤드가 가장 많습니다. [1]

1. **Secret, Route, OAuthClient** 오브젝트와 같은 민감한 리소스는 메타데이터 수준 이상으로는 기록되지 않습니다. 클러스터가 **OpenShift Container Platform 4.5**에서 업그레이드된 경우 **OAuth** 토큰은 해당 오브젝트 이름에 보안 정보가 포함될 수 있으므로 전혀 기록되지 않습니다.

기본적으로 OpenShift Container Platform에서는 Default 감사 로그 프로필을 사용합니다. 요청 본문까지 기록하는 다른 감사 정책 프로필을 사용할 수도 있지만 늘어나는 리소스 사용량(CPU, 메모리, I/O)을 알고 있어야 합니다.

8.2. 감사 로그 정책 구성

API 서버로 들어오는 요청을 기록할 때 사용할 감사 로그 정책을 구성할 수 있습니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. APIServer 리소스를 편집합니다.

```
$ oc edit apiserver cluster
```

2. spec.audit.profile 필드를 업데이트합니다.

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: WriteRequestBodies 1
```

1 Default, WriteRequestBodies 또는 AllRequestBodies로 설정합니다. 기본 프로필은 Default입니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

4. Kubernetes API 서버 Pod의 새 버전이 출시되었는지 확인합니다. 이 작업에는 몇 분 정도 걸립니다.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Kubernetes API 서버의 NodeInstallerProgressing 상태 조건을 검토하여 모든 노드가 최신 버전인지 확인합니다. 업데이트가 성공적으로 실행되면 출력에 AllNodesAtLatestRevision이 표시됩니다.

```
AllNodesAtLatestRevision
3 nodes are at revision 12 1
```

1

이 예에서 최신 버전 번호는 **12** 입니다.

출력에 다음 중 하나와 유사한 메시지가 표시되면 업데이트가 아직 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

9장. TLS 보안 프로필 설정

TLS 보안 프로필은 서버가 서버에 연결할 때 클라이언트가 사용할 수 있는 암호를 규제하는 방법을 제공합니다. 이렇게 하면 OpenShift Container Platform 구성 요소에서 알려진 비보안 프로토콜, 암호 또는 알고리즘을 허용하지 않는 암호화 라이브러리를 사용할 수 있습니다.

클러스터 관리자는 다음 구성 요소 각각에 사용할 TLS 보안 프로필을 선택할 수 있습니다.

- Ingress 컨트롤러
- 컨트롤 플레인

  여기에는 Kubernetes API 서버, OpenShift API 서버, OpenShift OAuth API 서버 및 OpenShift OAuth 서버가 포함됩니다.

9.1. TLS 보안 프로필 이해

TLS(Transport Layer Security) 보안 프로필을 사용하여 다양한 OpenShift Container Platform 구성 요소에 필요한 TLS 암호를 정의할 수 있습니다. OpenShift Container Platform TLS 보안 프로필은 Mozilla 권장 구성을 기반으로 합니다.

각 구성 요소에 대해 다음 TLS 보안 프로필 중 하나를 지정할 수 있습니다.

표 9.1. TLS 보안 프로필

Profile	설명
Old	이 프로필은 레거시 클라이언트 또는 라이브러리와 함께 사용하기 위한 것입니다. 프로필은 이전 버전과의 호환성 권장 구성을 기반으로 합니다. **Old** 프로파일에는 최소 TLS 버전 1.0이 필요합니다. **참고** Ingress 컨트롤러의 경우 최소 TLS 버전이 1.0에서 1.1로 변환됩니다.

Profile	설명
Intermediate	이 프로필은 대부분의 클라이언트에서 권장되는 구성입니다. Ingress 컨트롤러 및 컨트롤 플레인의 기본 TLS 보안 프로필입니다. 프로필은 중간 호환성 권장 구성을 기반으로 합니다. **Intermediate** 프로필에는 최소 TLS 버전이 1.2가 필요합니다.
Modern	이 프로필은 이전 버전과의 호환성이 필요하지 않은 최신 클라이언트와 사용하기 위한 것입니다. 이 프로필은 최신 호환성 권장 구성을 기반으로 합니다. **Modern** 프로필에는 최소 TLS 버전 1.3이 필요합니다. **참고** OpenShift Container Platform 4.6, 4.7, 4.8에서는 **Modern** 프로필이 지원되지 않습니다. 선택하면 **Intermediate** 프로필이 활성화됩니다. **중요** 현재 **Modern** 프로필은 지원되지 않습니다.
사용자 지정	이 프로필을 사용하면 사용할 TLS 버전과 암호를 정의할 수 있습니다. 주의 **Custom** 프로파일을 사용할 때는 잘못된 구성으로 인해 문제가 발생할 수 있으므로 주의해야 합니다. **참고** OpenShift Container Platform 라우터를 사용하면 Red Hat에서 배포한 OpenSSL 기본 TLS **1.3** 암호화 제품군 세트를 사용할 수 있습니다. OpenShift Container Platform 4.6, 4.7, 4.8에서는 TLS **1.3 연결 및 암호 제품군을 사용할 수 없습니다.**

참고

미리 정의된 프로파일 유형 중 하나를 사용하는 경우 유효한 프로파일 구성은 릴리스마다 변경될 수 있습니다. 예를 들어 릴리스 **X.Y.Z**에 배포된 중간 프로필을 사용하는 사양이 있는 경우 릴리스 **X.Y.Z+1**로 업그레이드하면 새 프로필 구성이 적용되어 롤아웃이 발생할 수 있습니다.

9.2. TLS 보안 프로필 세부 정보 보기

다음 각 구성 요소에 대해 사전 정의된 **TLS** 보안 프로필의 최소 **TLS** 버전 및 암호를 볼 수 있습니다. **Ingress** 컨트롤러 및 컨트롤 플레인입니다.

중요

프로필에 대한 최소 **TLS** 버전 및 암호 목록의 효과적인 구성은 구성 요소마다 다를 수 있습니다.

절차

- 특정 **TLS** 보안 프로필의 세부 정보를 확인합니다.

```
$ oc explain <component>.spec.tlsSecurityProfile.<profile> 1
```

1

<component> 에 대해 **ingresscontroller** 또는 **apiserver** 를 지정합니다. **<profile>**에 대해 **old, intermediate, custom**을 지정합니다.

예를 들어 컨트롤 플레인의 **intermediate** 프로필에 포함된 암호를 확인하려면 다음을 수행합니다.

```
$ oc explain apiserver.spec.tlsSecurityProfile.intermediate
```

출력 예

```
KIND:     APIServer
VERSION:  config.openshift.io/v1

DESCRIPTION:
```

```
    intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28reco
mmended.29
    and looks like this (yaml):
    ciphers: - TLS_AES_128_GCM_SHA256 - TLS_AES_256_GCM_SHA384 -
    TLS_CHACHA20_POLY1305_SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256 -
    ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES256-GCM-SHA384 -
    ECDHE-RSA-AES256-GCM-SHA384 - ECDHE-ECDSA-CHACHA20-POLY1305 -
    ECDHE-RSA-CHACHA20-POLY1305 - DHE-RSA-AES128-GCM-SHA256 -
    DHE-RSA-AES256-GCM-SHA384 minTLSVersion: TLSv1.2
```

- 구성 요소의 **tlsSecurityProfile** 필드에 대한 모든 세부 정보를 확인합니다.

```
$ oc explain <component>.spec.tlsSecurityProfile 1
```

1 **<component>** 에 대해 **ingresscontroller** 또는 **apiserver** 를 지정합니다.

예를 들어 **Ingress** 컨트롤러의 **tlsSecurityProfile** 필드의 모든 세부 정보를 확인하려면 다음을 수행합니다.

```
$ oc explain ingresscontroller.spec.tlsSecurityProfile
```

출력 예

```
KIND:     IngressController
VERSION:  operator.openshift.io/v1

RESOURCE: tlsSecurityProfile <Object>

DESCRIPTION:
     ...

FIELDS:
   custom <>
     custom is a user-defined TLS security profile. Be extremely careful using a
     custom profile as invalid configurations can be catastrophic. An example
     custom profile looks like this:
     ciphers: - ECDHE-ECDSA-CHACHA20-POLY1305 - ECDHE-RSA-CHACHA20-
POLY1305 -
```

```
    ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256
minTLSVersion:
    TLSv1.1

  intermediate <>
    intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28reco
mmended.29
    and looks like this (yaml):
    ... 1

  modern <>
    modern is a TLS security profile based on:
    https://wiki.mozilla.org/Security/Server_Side_TLS#Modern_compatibility and
    looks like this (yaml):
    ... 2
    NOTE: Currently unsupported.

  old <>
    old is a TLS security profile based on:
    https://wiki.mozilla.org/Security/Server_Side_TLS#Old_backward_compatibility
    and looks like this (yaml):
    ... 3

  type <string>
    ...
```

1 여기에서는 **intermediate** 프로필의 암호 및 최소 버전을 나열합니다.

2 **modern** 프로필의 암호 및 최소 버전을 나열합니다.

3 **old** 프로필의 암호 및 최소 버전을 여기에서 나열합니다.

9.3. INGRESS 컨트롤러의 TLS 보안 프로필 구성

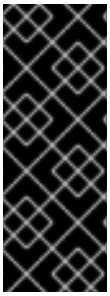
Ingress 컨트롤러에 대한 **TLS** 보안 프로필을 구성하려면 **IngressController CR**(사용자 정의 리소스)을 편집하여 사전 정의된 또는 사용자 지정 **TLS** 보안 프로필을 지정합니다. **TLS** 보안 프로필이 구성되지 않은 경우 기본값은 **API** 서버에 설정된 **TLS** 보안 프로필을 기반으로 합니다.

Old TLS 보안 프로파일을 구성하는 샘플 IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS 보안 프로필은 Ingress 컨트롤러의 TLS 연결에 대한 최소 TLS 버전과 TLS 암호를 정의합니다.

Status.Tls Profile 아래의 IngressController CR(사용자 정의 리소스) 및 Spec.Tls Security Profile 아래 구성된 TLS 보안 프로필에서 구성된 TLS 보안 프로필의 암호 및 최소 TLS 버전을 확인할 수 있습니다. Custom TLS 보안 프로필의 경우 특정 암호 및 최소 TLS 버전이 두 매개변수 아래에 나열됩니다.



중요

HAProxy Ingress 컨트롤러 이미지는 TLS 1.3을 지원하지 않으며 Modern 프로파일에는 TLS 1.3이 필요하므로 이는 지원되지 않습니다. Ingress Operator는 Modern 프로파일을 Intermediate로 변환합니다. 또한, Ingress Operator는 Old 또는 Custom 프로파일의 TLS 1.0을 1.1로 변환하고 Custom 프로파일의 TLS 1.3을 1.2로 변환합니다.

사전 요구 사항

- **cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.**

절차

1. **openshift-ingress-operator 프로젝트에서 IngressController CR을 편집하여 TLS 보안 프로필을 구성합니다.**

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. **spec.tlsSecurityProfile 필드를 추가합니다.**

Custom 프로필에 대한 IngressController CR 샘플

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
...
```

(1) TLS 보안 프로필 유형(Old,Intermediate 또는 Custom)을 지정합니다. 기본값은 Intermediate입니다.

(2) 선택한 유형의 적절한 필드를 지정합니다.

- old: {}
- intermediate: {}
- custom:

(3) custom 유형의 경우 TLS 암호화 목록 및 최소 허용된 TLS 버전을 지정합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

검증

- IngressController CR에 프로파일이 설정되어 있는지 확인합니다.

```
$ oc describe IngressController default -n openshift-ingress-operator
```

출력 예

```
Name:         default
Namespace:    openshift-ingress-operator
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

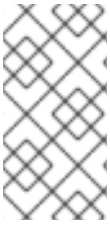
9.4. 컨트롤 플레인의 TLS 보안 프로필 구성

컨트롤 플레인에 대한 TLS 보안 프로필을 구성하려면 APIServer CR(사용자 정의 리소스)을 편집하여 사전 정의된 또는 사용자 지정 TLS 보안 프로필을 지정합니다. APIServer CR에서 TLS 보안 프로필을 설정하면 다음 컨트롤 플레인 구성 요소로 설정이 전파됩니다.

- Kubernetes API 서버
- OpenShift API 서버

- OpenShift OAuth API 서버
- OpenShift OAuth 서버

TLS 보안 프로필이 구성되지 않은 경우 기본 TLS 보안 프로필은 Intermediate입니다.



참고

Ingress 컨트롤러의 기본 TLS 보안 프로필은 API 서버에 설정된 TLS 보안 프로필을 기반으로 합니다.

Old TLS 보안 프로파일을 구성하는 샘플 APIServer CR

```
apiVersion: config.openshift.io/v1
kind: APIServer
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS 보안 프로필은 컨트롤 플레인 구성 요소와 통신하는 데 필요한 최소 TLS 버전 및 TLS 암호를 정의합니다.

Spec.Tls Security Profile의 APIServer CR(사용자 정의 리소스)에서 구성된 TLS 보안 프로필을 확인할 수 있습니다. Custom TLS 보안 프로필의 경우 특정 암호 및 최소 TLS 버전이 나열됩니다.

참고

컨트롤 플레인은 TLS 1.3을 최소 TLS 버전으로 지원하지 않습니다. TLS 1.3이 필요하므로 Modern 프로파일은 지원되지 않습니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. 기본 APIServer CR을 편집하여 TLS 보안 프로필을 구성합니다.

```
$ oc edit APIServer cluster
```

2. spec.tlsSecurityProfile 필드를 추가합니다.

Custom 프로파일의 APIServer CR 샘플

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
```

(1) TLS 보안 프로필 유형(Old,Intermediate 또는 Custom)을 지정합니다. 기본값은 Intermediate입니다.

(2) 선택한 유형의 적절한 필드를 지정합니다.

- old: {}

- intermediate: {}

- custom:

3 custom 유형의 경우 TLS 암호화 목록 및 최소 허용된 TLS 버전을 지정합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

검증

- TLS 보안 프로필이 APIServer CR에 설정되어 있는지 확인합니다.

```
$ oc describe apiserver cluster
```

출력 예

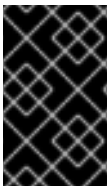
```
Name:         cluster
Namespace:
 ...
API Version:  config.openshift.io/v1
Kind:         APIServer
 ...
Spec:
  Audit:
    Profile:  Default
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

10장. SECCOMP 프로필 구성

OpenShift Container Platform 컨테이너 또는 Pod는 하나 이상의 잘 정의된 작업을 수행하는 단일 애플리케이션을 실행합니다. 애플리케이션에는 일반적으로 기본 운영 체제 커널 API의 작은 하위 집합만 필요합니다. seccomp 보안 컴퓨팅 모드는 사용 가능한 시스템 호출의 하위 집합만 호출하도록 컨테이너에서 실행되는 프로세스를 제한하는 데 사용할 수 있는 Linux 커널 기능입니다. 이러한 시스템 호출은 컨테이너 또는 Pod에 적용되는 프로필을 생성하여 구성할 수 있습니다. seccomp 프로필은 디스크에 JSON 파일로 저장됩니다.

중요

OpenShift 워크로드는 seccomp 프로필이 적용되지 않고 기본적으로 제한 없이 실행됩니다.



중요

seccomp 프로필은 권한 있는 컨테이너에 적용할 수 없습니다.

10.1. 모든 POD에 대한 기본 SECCOMP 프로필 활성화

OpenShift Container Platform에는 런타임/기본값으로 참조되는 기본 seccomp 프로필이 제공됩니다. 사용자 지정 SCC(보안 컨텍스트 제약 조건)를 생성하여 Pod 또는 컨테이너 워크로드에 대한 기본 seccomp 프로필을 활성화할 수 있습니다.

참고

사용자 지정 SCC를 생성해야 합니다. 기본 SCC를 편집하지 마십시오. 기본 SCC를 편집하면 일부 플랫폼 Pod 배포 또는 OpenShift Container Platform이 업그레이드되면 문제가 발생할 수 있습니다. 자세한 내용은 "기본 보안 컨텍스트 제약 조건" 섹션을 참조하십시오.

다음 단계에 따라 모든 Pod에 대한 기본 seccomp 프로필을 활성화합니다.

1. 사용 가능한 제한된 SCC를 yaml 파일로 내보냅니다.

```
$ oc get scc restricted -o yaml > restricted-seccomp.yaml
```

2. 생성된 **restricted SCC yaml** 파일을 편집합니다.

```
$ vi restricted-seccomp.yaml
```

3. 다음 예제와 같이 를 업데이트합니다.

```
kind: SecurityContextConstraints
metadata:
  name: restricted (1)
<..snip..>
seccompProfiles: (2)
- runtime/default (3)
```

(1) **restricted-seccomp**로 변경

(2) **seccompProfile**을 추가합니다.

(3) 추가 **-** 런타임/기본

4. 사용자 지정 **SCC**를 생성합니다.

```
$ oc create -f restricted-seccomp.yaml
```

예상 출력

```
securitycontextconstraints.security.openshift.io/restricted-seccomp created
```

5. **ServiceAccount**에 사용자 정의 **SCC**를 추가합니다.

```
$ oc adm policy add-scc-to-user restricted-seccomp -z default
```

-

참고

기본 서비스 계정은 사용자가 다른 서비스를 구성하지 않는 한 적용되는 **ServiceAccount**입니다. **OpenShift Container Platform**은 **SCC**의 정보를 기반으로 **pod**의 **seccomp** 프로필을 구성합니다.

예상 출력

```
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:restricted-seccomp added: "default"
```

OpenShift Container Platform 4.6에서 **pod** 주석 **seccomp.security.alpha.kubernetes.io/pod: runtime/default** 및 **container.seccomp.security.alpha.kubernetes.io/<container_name>: runtime/default**는 더 이상 사용되지 않습니다.

10.2. 사용자 정의 SECCOMP 프로필 구성

애플리케이션 요구 사항에 따라 필터를 업데이트할 수 있는 사용자 지정 **seccomp** 프로필을 구성할 수 있습니다. 이를 통해 클러스터 관리자는 **OpenShift Container Platform**에서 실행되는 워크로드의 보안을 보다 효과적으로 제어할 수 있습니다.

10.2.1. 사용자 정의 seccomp 프로필 설정

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- 사용자 정의 **SCC**(보안 컨텍스트 제약 조건)를 생성했습니다. 자세한 내용은 "추가 리소스"를 참조하십시오.
- 사용자 지정 **seccomp** 프로필을 생성했습니다.

프로세스

1. Machine Config를 사용하여 사용자 정의 seccomp 프로필을 /var/lib/kubelet/seccomp/<custom-name>.json에 업로드합니다. 자세한 단계는 "추가 리소스"를 참조하십시오.

2. 생성된 사용자 지정 seccomp 프로필에 대한 참조를 제공하여 사용자 정의 SCC를 업데이트합니다.

```
seccompProfiles:
- localhost/<custom-name>.json 1
```

1. 사용자 지정 seccomp 프로필의 이름을 제공합니다.

10.2.2. 워크로드에 사용자 정의 seccomp 프로필 적용

사전 요구 사항

- 클러스터 관리자가 사용자 지정 seccomp 프로필을 설정했습니다. 자세한 내용은 "사용자 지정 seccomp 프로필 설정"을 참조하십시오.

프로세스

- securityContext.seccompProfile.type 필드를 다음과 같이 설정하여 워크로드에 seccomp 프로필을 적용합니다.

예제

```
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: <custom-name>.json 1
```

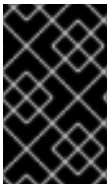
1. 사용자 지정 seccomp 프로필의 이름을 제공합니다.

또는 **Pod** 주석 **seccomp.security.alpha.kubernetes.io/pod: localhost/<custom-name>.json**을 사용할 수 있습니다. 그러나 이 방법은 **OpenShift Container Platform 4.6**에서 더 이상 사용되지 않습니다.

배포하는 동안 승인 컨트롤러는 다음을 확인합니다.

- 사용자 역할에서 허용하는 현재 **SCC**에 대한 주석입니다.
- **seccomp** 프로필을 포함하는 **SCC**를 **pod**에 사용할 수 있습니다.

Pod에 **SCC**가 허용되면 **kubelet**은 지정된 **seccomp** 프로필을 사용하여 **Pod**를 실행합니다.



중요

seccomp 프로필이 모든 작업자 노드에 배포되었는지 확인합니다.

참고

사용자 정의 **SCC**에는 **CAP_NET_ADMIN** 허용과 같은 **Pod**에 필요한 다른 조건을 충족하거나 **Pod**에 자동으로 할당되는 적절한 우선순위가 있어야 합니다.

10.3. 추가 리소스

- 보안 컨텍스트 제약 조건 관리
- 설치 후 시스템 구성 작업

11장. 추가 호스트에서 JAVASCRIPT를 기반으로 API 서버에 액세스하도록 허용

11.1. 추가 호스트에서 JAVASCRIPT를 기반으로 API 서버에 액세스하도록 허용

기본 OpenShift Container Platform 구성에서는 OpenShift 웹 콘솔에서만 요청을 API 서버로 보낼 수 있습니다.

다른 호스트 이름을 사용하여 JavaScript 애플리케이션에서 API 서버 또는 OAuth 서버에 액세스해야 하는 경우 허용할 추가 호스트 이름을 구성할 수 있습니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. APIServer 리소스를 편집합니다.

```
$ oc edit apiserver.config.openshift.io cluster
```

2. spec 섹션 아래에 additionalCORSAllowedOrigins 필드를 추가하고 하나 이상의 추가 호스트 이름을 지정합니다.

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-07-11T17:35:37Z"
  generation: 1
  name: cluster
  resourceVersion: "907"
  selfLink: /apis/config.openshift.io/v1/apiservers/cluster
  uid: 4b45a8dd-a402-11e9-91ec-0219944e0696
spec:
  additionalCORSAllowedOrigins:
  - (?i)//my\.subdomain\.domain\.com(:|\z) 1
```

1

호스트 이름은 API 서버 및 OAuth 서버에 대한 HTTP 요청의 CORS 헤더와 일치하는 Golang 정규식으로 지정됩니다.

참고

이 예에서는 다음 구문을 사용합니다.

- **(?i)**는 대소문자를 구분하지 않습니다.
- //는 도메인 시작 부분에 고정되고 **http:** 또는 **https:** 다음의 이중 슬래시와 일치합니다.
- \.은 도메인 이름에서 점을 이스케이프합니다.
- **(:|\z)**는 도메인 이름 **(\z)**의 끝 또는 포트 구분 기호 **(:)**과 일치하는지 확인합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

12장. ETCD 데이터 암호화

12.1. ETCD 암호화 정보

기본적으로 etcd 데이터는 OpenShift Container Platform에서 암호화되지 않습니다. 클러스터에 etcd 암호화를 사용하여 추가 데이터 보안 계층을 제공할 수 있습니다. 예를 들어 etcd 백업이 잘못된 당사자에게 노출되는 경우 중요한 데이터의 손실을 방지할 수 있습니다.

etcd 암호화를 활성화하면 다음 OpenShift API 서버 및 쿠버네티스 API 서버 리소스가 암호화됩니다.

- 보안
- 구성 맵
- 라우트
- OAuth 액세스 토큰
- OAuth 승인 토큰

etcd 암호화를 활성화하면 암호화 키가 생성됩니다. 이 키는 매주 순환됩니다. etcd 백업에서 복원하려면 이 키가 있어야 합니다.

참고

etcd 암호화는 키가 아닌 값만 암호화합니다. 즉, 리소스 유형, 네임 스페이스 및 개체 이름은 암호화되지 않습니다.

12.2. ETCD 암호화 활성화

etcd 암호화를 활성화하여 클러스터에서 중요한 리소스를 암호화할 수 있습니다.

주의

초기 암호화 프로세스가 완료될 때까지 **etcd**의 백업을 수행하지 않는 것이 좋습니다. 암호화 프로세스가 완료되지 않은 경우 백업은 부분적으로만 암호화될 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. **APIServer** 오브젝트를 수정합니다.

```
$ oc edit apiserver
```

2. 암호화 필드 유형을 **aescbc**로 설정합니다.

```
spec:
  encryption:
    type: aescbc 1
```

1 **aescbc** 유형은 **PKCS# 7** 패딩 및 **32**바이트 키가 있는 **AES-CBC**가 암호화를 수행하는 데 사용됨을 나타냅니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

암호화 프로세스가 시작됩니다. 클러스터 크기에 따라 이 프로세스를 완료하는 데 **20**분 이상 걸릴 수 있습니다.

4. **etcd** 암호화에 성공했는지 확인합니다.

a. **OpenShift API** 서버의 **Encrypted** 상태 조건을 검토하여 해당 리소스가 성공적으로 암

호화되었는지 확인합니다.

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호화에 성공하면 출력에 **EncryptionCompleted**가 표시됩니다.

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io,
oauthaccesstokens.oauth.openshift.io, oauthauthorizetokens.oauth.openshift.io
```

출력에 **EncryptionInProgress**가 표시되면 암호화가 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

b.

쿠버네티스 **API** 서버의 **Encrypted** 상태 조건을 검토하여 해당 리소스가 성공적으로 암호화되었는지 확인합니다.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호화에 성공하면 출력에 **EncryptionCompleted**가 표시됩니다.

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

출력에 **EncryptionInProgress**가 표시되면 암호화가 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

12.3. ETCD 암호화 비활성화

클러스터에서 **etcd** 데이터의 암호화를 비활성화할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. APIServer 오브젝트를 수정합니다.

```
$ oc edit apiserver
```

2. 암호화 필드 유형을 identity로 설정합니다.

```
spec:
  encryption:
    type: identity 1
```

1 identity 유형이 기본값이며, 이는 암호화가 수행되지 않음을 의미합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

암호 해독 프로세스가 시작됩니다. 클러스터 크기에 따라 이 프로세스를 완료하는 데 20분 이상 걸릴 수 있습니다.

4. etcd 암호 해독에 성공했는지 확인합니다.

a. OpenShift API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스의 암호가 성공적으로 해독되었는지 확인합니다.

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호 해독에 성공하면 출력에 DecryptionCompleted가 표시됩니다.

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

출력에 DecryptionInProgress가 표시되면 암호 해독이 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

b. 쿠버네티스 API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스의 암호가 성공적으로 해독되었는지 확인합니다.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호 해독에 성공하면 출력에 **DecryptionCompleted**가 표시됩니다.

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

출력에 **DecryptionInProgress**가 표시되면 암호 해독이 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

13장. POD에서 취약점 스캔

Red Hat Quay Container Security Operator를 사용하면 OpenShift Container Platform 웹 콘솔에서 클러스터의 활성 Pod에 사용되는 컨테이너 이미지의 취약점 검사 결과에 액세스할 수 있습니다. Red Hat Quay Container Security Operator:

- 모든 네임스페이스 또는 지정된 네임스페이스에서 Pod와 관련된 컨테이너 감시
- 이미지의 레지스트리에서 이미지 스캔(예: Clair 스캔을 사용하는 Quay.io 또는 Red Hat Quay 레지스트리)이 실행 중인 경우 컨테이너가 있는 컨테이너 레지스트리에서 취약점 정보를 조회합니다.
- Kubernetes API의 ImageManifestVuln 오브젝트를 통해 취약점 노출

여기에서 지침을 사용하여 Red Hat Quay Container Security Operator가 openshift-operators 네임스페이스에 설치되므로 OpenShift 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.

13.1. RED HAT QUAY CONTAINER SECURITY OPERATOR 실행

여기에 설명된 대로 Operator Hub에서 해당 Operator를 선택하고 설치하여 OpenShift Container Platform 웹 콘솔에서 Red Hat Quay Container Security Operator를 시작할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터에 대한 관리자 권한 보유
- 클러스터에서 실행 중인 Red Hat Quay 또는 Quay.io 레지스트리의 컨테이너 보유

프로세스

1. Operators → OperatorHub로 이동하여 보안을 선택합니다.
2. 컨테이너 보안 Operator를 선택한 후 설치를 선택하여 Operator 서브스크립션 생성 페이지로 이동합니다.

3. 설정을 확인합니다. 기본적으로 모든 네임스페이스 및 자동 승인 전략이 선택됩니다.

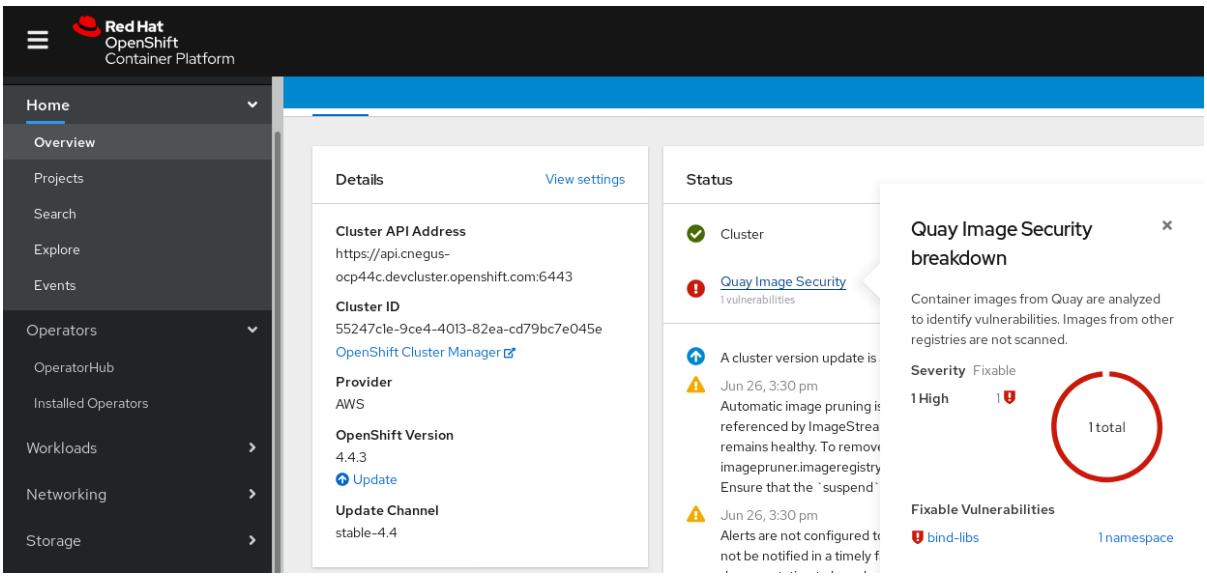
4. 설치를 선택합니다. 컨테이너 보안 Operator가 설치된 Operator 화면에 잠시 후에 나타납니다.

5. 선택적으로 Red Hat Quay Container Security Operator에 사용자 정의 인증서를 추가할 수 있습니다. 이 예에서는 현재 디렉터리에 quay.crt라는 인증서를 생성합니다. 그런 다음 다음 명령을 실행하여 Red Hat Quay Container Security Operator에 인증서를 추가합니다.

```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -
n openshift-operators
```

6. 사용자 정의 인증서를 추가한 경우 새 인증서가 적용되도록 Operator Pod를 다시 시작합니다.

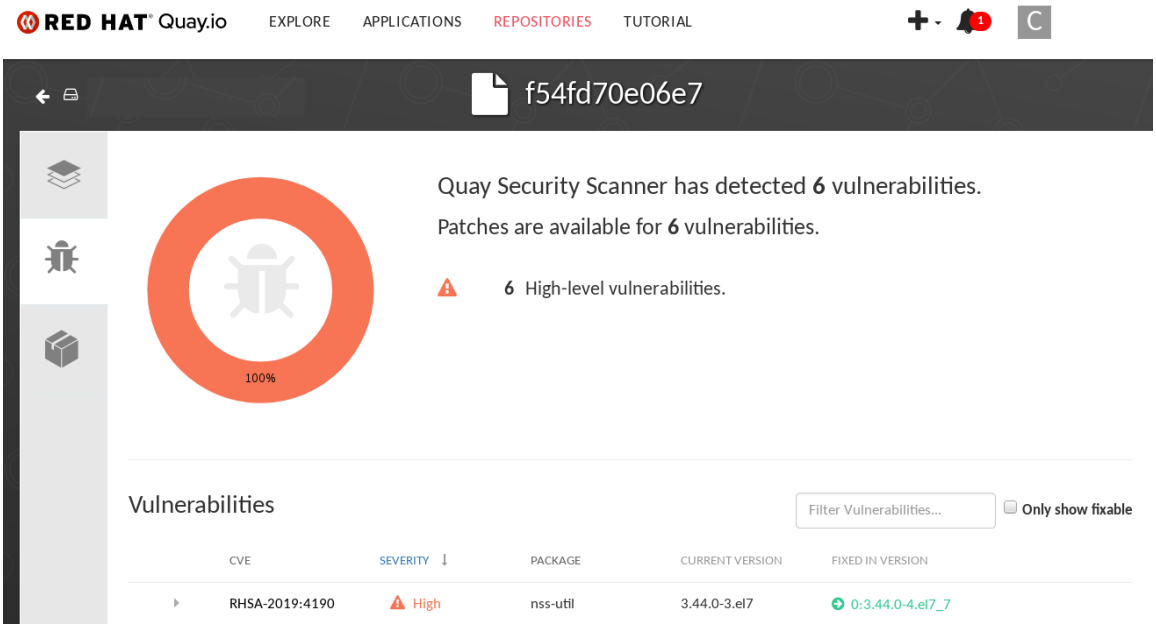
7. OpenShift 대시보드(홈 → 개요)를 엽니다. Quay Image Security의 링크가 지금까지 발견된 취약점 수와 함께 상태 섹션에 표시됩니다. 다음 그림과 같이 링크를 선택하여 Quay Image Security 분석을 확인합니다.



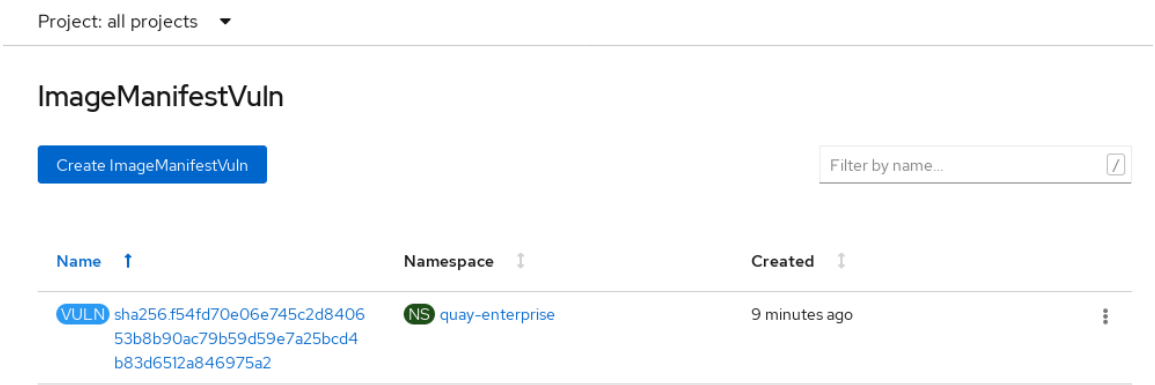
8. 이 시점에서 두 가지 중 하나를 수행하여 탐지된 취약점을 추적할 수 있습니다.

- 취약점의 링크를 선택합니다. 컨테이너가 제공된 컨테이너 레지스트리로 이동하여 취약점에 관한 정보를 볼 수 있습니다. 다음 그림은 Quay.io 레지스트리에서 발견된 취약점의 예

를 보여줍니다.



- 네임스페이스 링크를 선택하여 **ImageManifestVuln** 화면으로 이동합니다. 여기서 선택한 이미지의 이름과 해당 이미지가 실행 중인 모든 네임스페이스를 볼 수 있습니다. 다음 그림은 취약한 특정 이미지가 **quay-enterprise** 네임스페이스에서 실행 중임을 나타냅니다.



이 시점에서 취약한 이미지, 해당 취약점을 수정하기 위해 수행해야 할 작업 및 이미지가 실행된 모든 네임스페이스를 알 수 있습니다. 따라서 다음을 수행할 수 있습니다.

- 이미지를 실행하는 모든 사람에게 취약점을 수정해야 함을 경고합니다.

- 이미지가 있는 Pod를 시작한 배포 또는 기타 오브젝트를 삭제하여 이미지 실행을 중지합니다.

Pod를 삭제하면 대시보드에서 취약점이 재설정되는 데 몇 분이 걸릴 수 있습니다.

13.2. CLI에서 이미지 취약점 쿼리

oc 명령을 사용하면 Red Hat Quay Container Security Operator에서 탐지한 취약점에 대한 정보를 표시할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 인스턴스에서 Red Hat Quay Container Security Operator 실행

프로세스

- 탐지된 컨테이너 이미지 취약점을 조회하려면 다음을 입력합니다.

```
$ oc get vuln --all-namespaces
```

출력 예

```
NAMESPACE    NAME           AGE
default      sha256.ca90...   6m56s
skynet       sha256.ca90...   9m37s
```

- 특정 취약점에 관한 세부 정보를 표시하려면 취약점 이름 및 해당 네임스페이스를 oc describe 명령에 제공합니다. 이 예에서는 이미지에 취약점이 있는 RPM 패키지가 포함된 활성 컨테이너를 보여줍니다.

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
```

출력 예

```
Name:         sha256.ac50e3752...
Namespace:    quay-enterprise
...
Spec:
  Features:
    Name:           nss-util
    Namespace Name:  centos:7
    Version:        3.44.0-3.el7
    Versionformat:   rpm
    Vulnerabilities:
      Description: Network Security Services (NSS) is a set of libraries...
```