Red Hat OpenShift Pipelines 1.15

Securing OpenShift Pipelines

OpenShift Pipelines의 보안 기능

Last Updated: 2024-09-02

Red Hat OpenShift Pipelines 1.15 Securing OpenShift Pipelines

OpenShift Pipelines의 보안 기능

법적 공지

초록

이 문서에서는 OpenShift Pipelines의 보안 기능에 대한 정보를 제공합니다.

차례

1장. OPENSHIFT PIPELINES 공급망 보안에 TEKTON 체인 사용 3
1.1. 주요 기능 3
1.2. TEKTON 체인 구성 3
1.3. TEKTON 체인에서 데이터에 서명하기 위한 보안 10
1.4. OCI 레지스트리에 인증 12
1.5. 추가 인증 없이 작업 실행 서명 생성 및 확인 13
1.6. TEKTON CHAIN을 사용하여 이미지와 검증에 서명 및 확인 15
1.7. 추가 리소스 17

2장. 소프트웨어 공급망 보안 요소를 보려면 웹 콘솔에서 OPENSHIFT PIPELINES 설정 18
2.1. 프로젝트 취약점을 보기 위해 OPENSHIFT PIPELINES 설정 18
2.2. SBOM을 다운로드하거나 볼 수 있도록 OPENSHIFT PIPELINES 설정 21
2.3. 추가 리소스 25

3장. POD의 보안 컨텍스트 구성 26
3.1. OPENSHIFT PIPELINES에서 생성하는 POD의 기본 및 최대 SCC 구성 26
3.2. 네임스페이스에서 POD의 SCC 구성 27
3.3. 사용자 정의 SCC 및 사용자 정의 서비스 계정을 사용하여 파이프라인 실행 및 작업 실행 27
3.4. 추가 리소스 29

4장. 이벤트 리스너로 WEBHOOK 보안 30
4.1. OPENSHIFT 경로와의 보안 연결 제공 30
4.2. 보안 HTTPS 연결을 사용하여 샘플 EVENTLISTENER 리소스 생성 31

5장. 보안을 사용하여 리포지터리로 파이프라인 인증 32
5.1. 사전 요구 사항 32
5.2. 서비스 계정을 사용하여 보안 제공 32
5.3. 작업 영역을 사용하여 보안 제공 41

6장. BUILDAH를 루트가 아닌 사용자로 사용하여 컨테이너 이미지 빌드 46
6.1. 사용자 네임스페이스를 구성하여 루트가 아닌 사용자로 BUILDAH 실행 46
6.2. 사용자 지정 SA 및 SCC를 정의하여 ROOT가 아닌 사용자로 BUILDAH 실행 47
6.3. 권한이 없는 빌드의 제한 54

1장. OPENSHIFT PIPELINES 공급망 보안에 TEKTON 체인 사용

Tekton 체인은 Kubernetes CRD(Custom Resource Definition) 컨트롤러입니다. 이를 사용하여 Red Hat OpenShift Pipelines를 사용하여 생성된 작업 및 파이프라인의 공급망 보안을 관리할 수 있습니다.

기본적으로 Tekton 체인은 OpenShift Container Platform 클러스터의 모든 작업 실행 실행을 관찰합니다. 작업이 완료되면 Tekton 체인은 작업 실행의 스냅샷을 사용합니다. 그런 다음 스냅샷을 하나 이상의 표준 페이로드 형식으로 변환하고 마지막으로 모든 아티팩트를 서명하고 저장합니다.

작업 실행에 대한 정보를 캡처하기 위해 Tekton 체인은 **Result** 오브젝트를 사용합니다. 오브젝트를 사용할 수 없는 경우 Tekton은 OCI 이미지의 URL과 인증된 다이제스트를 체인합니다.

1.1. 주요 기능

- **cosign** 및 **skopeo** 와 같은 툴을 통해 생성된 암호화 키를 사용하여 작업 실행, 작업 실행 결과, OCI 레지스트리 이미지에 서명할 수 있습니다.
- 인증 형식(예: **in-to-to**)을 사용할 수 있습니다.
- OCI 리포지토리를 스토리지 백엔드로 사용하여 서명 및 서명된 아티팩트를 안전하게 저장할 수 있습니다.

1.2. TEKTON 체인 구성

Red Hat OpenShift Pipelines Operator는 기본적으로 Tekton 체인을 설치합니다. **TektonConfig** 사용자 정의 리소스를 수정하여 Tekton 체인을 구성할 수 있습니다. Operator는 이 사용자 정의 리소스에서 변경한 사항을 자동으로 적용합니다.

사용자 정의 리소스를 편집하려면 다음 명령을 사용합니다.

```
$ oc edit TektonConfig config
```

사용자 정의 리소스에는 **체인 배열이 포함됩니다**. 다음 예와 같이 지원되는 모든 구성 매개변수를 이 배열에 추가할 수 있습니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  addon: {}
  chain:
    artifacts.taskrun.format: tekton
  config: {}
```

1.2.1. Tekton 체인 구성에 지원되는 매개변수

클러스터 관리자는 지원되는 다양한 매개변수 키와 값을 사용하여 작업 실행, OCI 이미지 및 스토리지에 대한 사양을 구성할 수 있습니다.

1.2.1.1. 작업 실행 아티팩트에 지원되는 매개변수

표 1.1. 체인 구성: 작업 실행 아티팩트에 대해 지원되는 매개변수

키	설명	지원되는 값	기본값
artifacts.taskrun.format	작업 실행 페이로드를 저장하는 형식입니다.	**in-toto**, **slsa/v1**	**In-toto**
artifacts.taskrun.storage	작업 실행 서명을 위한 스토리지 백엔드입니다. **"tekton,oci"** 와 같이 쉼표로 구분된 목록으로 여러 백엔드를 지정할 수 있습니다. 작업 실행 아티팩트 저장을 비활성화하려면 빈 문자열 **""** 를 제공합니다.	**Tekton**,**oci**,**gcs**,**docdb**,**grafeas**	**Tekton**
artifacts.taskrun.signer	작업 실행 페이로드에 서명하기 위한 서명 백엔드입니다.	**x509**,**kms**	**x509**

참고

slsa/v1 은 이전 버전과의 호환성을 위해 **In-to** 의 별칭입니다.

1.2.1.2. 파이프라인 실행 아티팩트에 지원되는 매개변수

표 1.2. 체인 구성: 파이프라인 실행 아티팩트에 대해 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
artifacts.pipelinerun.format	파이프라인 실행 페이로드를 저장하는 형식입니다.	**in-toto**, **slsa/v1**	**In-toto**
artifacts.pipelinerun.storage	파이프라인 실행 서명을 저장하기 위한 스토리지 백엔드입니다. **"tekton,oci"** 와 같이 쉼표로 구분된 목록으로 여러 백엔드를 지정할 수 있습니다. 파이프라인 실행 아티팩트 저장을 비활성화하려면 빈 문자열 **""** 를 제공합니다.	**Tekton**,**oci**,**gcs**,**docdb**,**grafeas**	**Tekton**
artifacts.pipelinerun.signer	파이프라인 실행 페이로드에 서명하기 위한 서명 백엔드입니다.	**x509**, **kms**	**x509**

매개변수	설명	지원되는 값	기본값
artifacts.pipelinerun.enable-deep-inspection	이 매개변수가 **true** 이면 Tekton 체인은 파이프라인 실행의 하위 작업 실행 결과를 기록합니다. 이 매개변수가 **false** 이면 Tekton 체인은 하위 작업 실행의 결과가 아니라 파이프라인 실행 결과를 기록합니다.	**"true", "false"**	**"false"**

참고

- **slsa/v1** 은 이전 버전과의 호환성을 위해 **In-to** 의 별칭입니다.
- **grafeas** 스토리지 백엔드의 경우 컨테이너 분석만 지원됩니다. Tekton 체인의 현재 버전에서는 **grafeas** 서버 주소를 구성할 수 없습니다.

1.2.1.3. OCI 아티팩트에 지원되는 매개변수

표 1.3. 체인 구성: OCI 아티팩트에 대해 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
artifacts.oci.format	OCI 페이로드를 저장하는 형식입니다.	**단순 서명**	**단순 서명**
artifacts.oci.storage	OCI 서명을 저장하기 위한 스토리지 백엔드입니다. **"oci,tekton"** 와 같이 쉼표로 구분된 목록으로 여러 백엔드를 지정할 수 있습니다. OCI 아티팩트 저장을 비활성화하려면 빈 문자열 **""** 를 제공합니다.	**Tekton**,**oci**,**gcs**,**docdb**,**grafeas**	**OCI**
artifacts.oci.signer	OCI 페이로드에 서명하기 위한 서명 백엔드입니다.	**x509**, **kms**	**x509**

1.2.1.4. KMS 서명자에서 지원되는 매개변수

표 1.4. 체인 설정: KMS 서명자에 대해 지원되는 매개변수

매개변수	설명	지원되는 값	기본값

매개변수	설명	지원되는 값	기본값
signers.kms.kmsref	**kms** 서명자에서 사용할 KMS 서비스에 대한 URI 참조입니다.	지원되는 스키마: **gcpkms://**, **awskms://**, **azurekms://**, **hashivault://**. 자세한 내용은 Sigstore 설명서의 KMS 지원을 참조하십시오.	

1.2.1.5. 스토리지에 지원되는 매개변수

표 1.5. 체인 구성: 스토리지에 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
storage.gcs.bucket	스토리지를 위한 GCS 버킷		
storage.oci.repository	OCI 서명 및 인증을 저장하기 위한 OCI 리포지토리입니다.	아티팩트 스토리지 백엔드 중 하나를 **oci** 로 구성하고 이 키를 정의하지 않으면 Tekton 체인은 저장된 OCI 아티팩트 자체와 함께 인증 정보를 저장합니다. 이 키를 정의하면 인증 정보는 OCI 아티팩트와 함께 저장되지 않으며 대신 지정된 위치에 저장됩니다. 자세한 내용은 cosign 문서를 참조하십시오.	
builder.id	인증 정보용으로 설정할 빌더 ID		**https://tekton.dev/chains/v2**

매개변수	설명	지원되는 값	기본값
builddefinition.buildtype	In-totestation의 빌드 유형입니다. 이 매개변수가 **https://tekton.dev/chains/v2/slsa** 이면 Tekton 체인은 SLSA v1.0 사양을 엄격하게 준수하여 인증 정보를 기록합니다. 이 매개변수가 **https://tekton.dev/chains/v2/slsa-tekton** 이면 Tekton 체인은 각 **TaskRun** 및 **PipelineRun** 오브젝트의 라벨 및 주석과 같은 추가 정보를 사용하여 In-toto Attestations를 기록하고 **resolvedDependencies** 아래의 **PipelineRun** 오브젝트에 각 작업을 추가합니다.	**https://tekton.dev/chains/v2/slsa**,**https://tekton.dev/chains/v2/slsa-tekton**	**https://tekton.dev/chains/v2/slsa**

docdb 스토리지 방법이 아티팩트에 해당하는 경우 docstore 스토리지 옵션을 구성합니다. go-cloud docstore URI 형식에 대한 자세한 내용은 docstore 패키지 설명서 를 참조하십시오. Red Hat OpenShift Pipelines는 다음 문서 서비스를 지원합니다.

- **Firestore**
- **dynamodb**

표 1.6. 체인 구성:docstore 스토리지에 대해 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
storage.docdb.url	**docstore** 컬렉션에 대한 go-cloud URI 참조입니다. 모든 아티팩트에 대해 **docdb** 스토리지 방법이 활성화된 경우 사용됩니다.	**Firestore://projects/[PROJECT]/databases/(default)/documents/[COLLECTION]?name_field=name**	

아티팩트에 대해 **grafeas** 스토리지 방법을 활성화하면 Grafeas 스토리지 옵션을 구성합니다. Grafeas 노트 및 발생에 대한 자세한 내용은 Grafeas 개념을 참조하십시오.

발생을 생성하기 위해 Red Hat OpenShift Pipelines는 먼저 발생을 연결하는 데 사용되는 노트를 생성해야 합니다. Red Hat OpenShift Pipelines는 두 가지 유형인 **ATTESTATION** Occurrence 및 **BUILD** Occurrence를 생성합니다.

Red Hat OpenShift Pipelines는 구성 가능한 **noteid** 를 노트 이름의 접두사로 사용합니다. **ATTESTATION** 노트에 대한 접미사 **-simplesigning** 과 **BUILD** note의 접미사 **-intoto** 를 추가합니다. **noteid** 필드가 구성되지 않은 경우 Red Hat OpenShift Pipelines는 **tekton-<NAMESPACE** >를 접두사로

사용합니다.

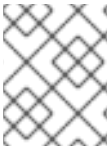
표 1.7. 체인 구성: Grafeas 스토리지에 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
storage.grafeas.projectid	발생을 저장하기 위한 Grafeas 서버가 있는 OpenShift Container Platform 프로젝트입니다.		
storage.grafeas.noteid	선택 사항: 생성된 모든 노트의 이름에 사용할 접두사입니다.	공백이 없는 문자열입니다.	
storage.grafeas.notehint	선택 사항: Grafeas **ATTESTATION** 참고 사항의 **human_readable_name** 필드입니다.		**이 인증 노트는 Tekton 체인에 의해 생성되었습니다.**

선택적으로 바이너리 투명성 증명의 추가 업로드를 활성화할 수 있습니다.

표 1.8. 체인 구성: 투명성 강화 스토리지를 위한 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
transparency.enabled	자동 바이너리 투명도 업로드를 활성화하거나 비활성화합니다.	**true,false,manual**	**false**
transparency.url	활성화된 경우 바이너리 투명도를 업로드하기 위한 URL입니다.		**https://rekor.sigstore.dev**



참고

transparency.enabled 를 **수동** 으로 설정하면 다음 주석이 있는 작업 실행 및 파이프라인 실행만 투명 로그에 업로드됩니다.

```
chains.tekton.dev/transparency-upload: "true"
```

x509 서명 백엔드를 구성하는 경우 Fulcio를 사용하여 키 없이 서명을 선택적으로 활성화할 수 있습니다.

표 1.9. 체인 구성: Fulcio를 사용한x509 키 없이 서명에 대해 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
signers.x509.fulcio.enabled	Fulcio에서 자동 인증서 요청 요청을 활성화하거나 비활성화합니다.	**true**,**false**	**false**
signers.x509.fulcio.address	활성화된 경우 인증서를 요청하는 Fulcio 주소입니다.		**https://v1.fulcio.sigstore.dev**
signers.x509.fulcio.issuer	예상되는 OIDC 발행자		**https://oauth2.sigstore.dev/auth**
signers.x509.fulcio.provider	ID 토큰을 요청할 공급자입니다.	**Google**,**spiffe**,**github**, **파일 시스템**	Red Hat OpenShift Pipelines는 모든 공급자를 사용하려고 합니다.
signers.x509.identity.token.file	ID 토큰이 포함된 파일의 경로입니다.		
signers.x509.tuf.mirror.url	TUF 서버의 URL입니다. **$TUF_URL/root.json**이 있어야 합니다.		**https://sigstore-tuf-root.storage.googleapis.com**

kms 서명 백엔드를 구성하는 경우 필요에 따라 OIDC 및 Spire를 포함한 KMS 구성을 설정합니다.

표 1.10. 체인 구성: KMS 서명에 지원되는 매개변수

매개변수	설명	지원되는 값	기본값
signers.kms.auth.address	KMS 서버의 URI(**VAULT_ADDR**값).		
signers.kms.auth.token	KMS 서버의 인증 토큰(**VAULT_TOKEN**값).		
signers.kms.auth.oidc.path	OIDC 인증 경로(예: Vault의 경우 **jwt**)입니다.		
signers.kms.auth.oidc.role	OIDC 인증의 역할입니다.		
signers.kms.auth.spire.sock	KMS 토큰에 대한 Spire 소켓의 URI입니다(예: **unix:///tmp/spire-agent/public/api.sock**).		

매개변수	설명	지원되는 값	기본값
signers.kms.auth.spire.audience	SVID를 Spire에서 요청하는 대상입니다.		

1.3. TEKTON 체인에서 데이터에 서명하기 위한 보안

클러스터 관리자는 키 쌍을 생성하고 Tekton Chains를 사용하여 Kubernetes 시크릿을 사용하여 아티팩트에 서명할 수 있습니다. Tekton 체인이 작동하려면 암호화된 키의 개인 키와 암호가 **openshift-pipelines** 네임스페이스에 **signing-secrets** 시크릿의 일부로 존재해야 합니다.

현재 Tekton 체인은 **x509** 및 **cosign** 서명 스키마를 지원합니다.

참고

지원되는 서명 체계 중 하나만 사용하십시오.

Tekton 체인과 **x509** 서명 스키마를 사용하려면 **ed25519** 또는 **ecdsa** 유형의 **x509.pem** 개인 키를 **signing-secrets** Kubernetes 시크릿에 저장합니다.

1.3.1. cosign을 사용하여 서명

cosign 툴을 사용하여 Tekton 체인에 공동 서명 스키마를 사용할 수 있습니다.

사전 요구 사항

- cosign 툴을 설치했습니다.

프로세스

1. 다음 명령을 실행하여 **cosign.key** 및 **cosign.pub** 키 쌍을 생성합니다.

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

   cosign은 암호를 입력하라는 메시지를 표시한 다음 Kubernetes 시크릿을 생성합니다.

2. 암호화된 **cosign.key** 개인 키와 **cosign.password** 암호 해독 암호를 **signing-secrets** Kubernetes 시크릿에 저장합니다. 개인 키가 **ENCRYPTED COSIGN PRIVATE KEY** 유형의 암호화된 PEM 파일로 저장되었는지 확인합니다.

1.3.2. skopeo를 사용하여 서명

skopeo 툴을 사용하여 키를 생성하고 Tekton 체인과 함께 **cosign** 서명 스키마에 사용할 수 있습니다.

사전 요구 사항

- skopeo 툴을 설치했습니다.

프로세스

1. 다음 명령을 실행하여 공개/개인 키 쌍을 생성합니다.

```
$ skopeo generate-sigstore-key --output-prefix <mykey> 1
```

1 & **lt;mykey** >를 선택한 키 이름으로 바꿉니다.

Skopeo는 개인 키에 대한 암호를 입력하라는 메시지를 표시한 다음 <mykey> **.private 및 <mykey** > **.pub 라는 키** 파일을 만듭니다.

2. 다음 명령을 실행하여 **base64** 툴을 사용하여 < **mykey>.pub** 파일을 인코딩합니다.

```
$ base64 -w 0 <mykey>.pub > b64.pub
```

3. 다음 명령을 실행하여 **base64** 툴을 사용하여 < **mykey>.private** 파일을 인코딩합니다.

```
$ base64 -w 0 <mykey>.private > b64.private
```

4. 다음 명령을 실행하여 **base64** 툴을 사용하여 passhprase를 인코딩합니다.

```
$ echo -n '<passphrase>' | base64 -w 0 > b64.passphrase 1
```

1 & **lt;passphrase&** gt;를 키 쌍에 사용한 암호로 바꿉니다.

5. 다음 명령을 실행하여 **openshift-pipelines** 네임스페이스에 **signing-secrets** 시크릿을 생성합니다.

```
$ oc create secret generic signing-secrets -n openshift-pipelines
```

6. 다음 명령을 실행하여 **signing-secrets** 시크릿을 편집합니다.

```
$ oc edit secret -n openshift-pipelines signing-secrets
```

다음과 같은 방식으로 시크릿 데이터에 인코딩된 키를 추가합니다.

```
apiVersion: v1
data:
  cosign.key: <Encoded <mykey>.private> 1
  cosign.password: <Encoded passphrase> 2
  cosign.pub: <Encoded <mykey>.pub> 3
immutable: true
kind: Secret
metadata:
  name: signing-secrets
# ...
type: Opaque
```

1 & **lt;Encoded <mykey>.private** >을 **b64.private** 파일의 내용으로 바꿉니다.

2 & **lt;Encoded passphrase&** gt;를 **b64.passphrase** 파일의 콘텐츠로 바꿉니다.

3 & **lt;Encoded <mykey>.pub** >를 **b64.pub** 파일의 내용으로 바꿉니다.

1.3.3. "비밀번호가 이미 있음" 오류 해결

signing-secret 시크릿이 이미 채워진 경우 이 보안을 생성하는 명령에서 다음 오류 메시지를 출력할 수 있습니다.

```
Error from server (AlreadyExists): secrets "signing-secrets" already exists
```

보안을 삭제하여 이 오류를 해결할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 **signing-secret** 시크릿을 삭제합니다.

```
$ oc delete secret signing-secrets -n openshift-pipelines
```

2. 키 쌍을 다시 생성하여 선호하는 서명 스키마를 사용하여 시크릿에 저장합니다.

1.4. OCI 레지스트리에 인증

클러스터 관리자는 OCI 레지스트리로 서명을 푸시하기 전에 레지스트리에 인증하도록 Tekton 체인을 구성해야 합니다. Tekton 체인 컨트롤러는 작업이 실행되는 동일한 서비스 계정을 사용합니다. 서명을 OCI 레지스트리로 내보내는 데 필요한 자격 증명으로 서비스 계정을 설정하려면 다음 단계를 수행합니다.

프로세스

1. Kubernetes 서비스 계정의 네임스페이스 및 이름을 설정합니다.

```
$ export NAMESPACE=<namespace> 1
$ export SERVICE_ACCOUNT_NAME=<service_account> 2
```

1. 서비스 계정과 연결된 네임스페이스입니다.
2. 서비스 계정의 이름입니다.

2. Kubernetes 시크릿을 생성합니다.

```
$ oc create secret registry-credentials \
  --from-file=.dockerconfigjson \ 1
  --type=kubernetes.io/dockerconfigjson \
  -n $NAMESPACE
```

1. Docker 구성 파일의 경로로 바꿉니다. 기본 경로는 **~/.docker/config.json** 입니다.

3. 서비스 계정에 시크릿에 대한 액세스 권한을 부여합니다.

```
$ oc patch serviceaccount $SERVICE_ACCOUNT_NAME \
  -p "{\"imagePullSecrets\": [{\"name\": \"registry-credentials\"}]}" -n $NAMESPACE
```

Red Hat OpenShift Pipelines가 모든 작업 실행에 할당하는 기본 **파이프라인** 서비스 계정을 패치하면 Red Hat OpenShift Pipelines Operator가 서비스 계정을 재정의합니다. 모범 사례로 다음 단계를 수행할 수 있습니다.

a. 사용자의 작업 실행에 할당할 별도의 서비스 계정을 생성합니다.

```
$ oc create serviceaccount <service_account_name>
```

b. 작업 실행 템플릿에서 **serviceaccountname** 필드의 값을 설정하여 서비스 계정을 작업 실행에 연결합니다.

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: build-push-task-run-2
spec:
  taskRunTemplate:
    serviceAccountName: build-bot (1)
  taskRef:
    name: build-push
...
```

(1) 새로 생성된 서비스 계정의 이름으로 바꿉니다.

1.5. 추가 인증 없이 작업 실행 서명 생성 및 확인

추가 인증과 함께 Tekton Chains를 사용하여 작업 실행 서명을 확인하려면 다음 작업을 수행합니다.

- 암호화된 x509 키 쌍을 생성하여 Kubernetes 시크릿으로 저장합니다.
- Tekton 체인 백엔드 스토리지를 구성합니다.
- 작업 실행을 생성하고 서명하고, 서명 및 페이로드를 작업 실행 자체에 주석으로 저장합니다.
- 서명된 작업 실행에서 서명 및 페이로드를 검색합니다.
- 작업 실행 서명을 확인합니다.

사전 요구 사항

다음 구성 요소가 클러스터에 설치되어 있는지 확인합니다.

- Red Hat OpenShift Pipelines Operator
- Tekton 체인
- cosign

프로세스

1. 암호화된 x509 키 쌍을 생성하여 Kubernetes 시크릿으로 저장합니다. 키 쌍을 생성하여 시크릿으로 저장하는 방법에 대한 자세한 내용은 " Tekton 체인에서 시크릿 서명"을 참조하십시오.
2. Tekton 체인 구성에서 OCI 스토리지를 비활성화하고 작업 실행 스토리지 및 형식을 **tekton** 로 설정합니다. **TektonConfig** 사용자 지정 리소스에서 다음 값을 설정합니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
```

```
metadata:
  name: config
spec:
# ...
    chain:
      artifacts.oci.storage: ""
      artifacts.taskrun.format: tekton
      artifacts.taskrun.storage: tekton
# ...
```

TektonConfig 사용자 지정 리소스를 사용하여 Tekton 체인 구성에 대한 자세한 내용은 " Tekton 체인 구성"을 참조하십시오.

3. Tekton 체인 컨트롤러를 다시 시작하여 수정된 구성이 적용되었는지 확인하려면 다음 명령을 입력합니다.

```
$ oc delete po -n openshift-pipelines -l app=tekton-chains-controller
```

4. 다음 명령을 입력하여 작업 실행을 생성합니다.

```
$ oc create -f
https://raw.githubusercontent.com/tektoncd/chains/main/examples/taskruns/task-output-
image.yaml 1
```

1 예제 URI를 작업 실행을 가리키는 URI 또는 파일 경로로 바꿉니다.

출력 예

```
taskrun.tekton.dev/build-push-run-output-image-qbjvh created
```

5. 다음 명령을 입력하여 단계의 상태를 확인합니다. 프로세스가 완료될 때까지 기다립니다.

```
$ tkn tr describe --last
```

출력 예

```
[...truncated output...]
NAME                          STATUS
· create-dir-builtimage-9467f   Completed
· git-source-sourcerepo-p2sk8   Completed
· build-and-push                Completed
· echo                        Completed
· image-digest-exporter-xlkn7   Completed
```

6. **base64** 로 인코딩된 주석으로 저장된 오브젝트에서 서명을 검색하려면 다음 명령을 입력합니다.

```
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\.tekton\.dev/signature-
taskrun-$TASKRUN_UID}" | base64 -d > sig
```

```
$ export TASKRUN_UID=$(tkn tr describe --last -o  jsonpath='{.metadata.uid}')
```

7. 생성한 공개 키를 사용하여 서명을 확인하려면 다음 명령을 입력합니다.

```
$ cosign verify-blob-attestation --insecure-ignore-tlog --key path/to/cosign.pub --signature sig --type slsaprovenance --check-claims=false /dev/null 1
```

1 **path/to/cosign.pub** 를 공개 키 파일의 경로 이름으로 바꿉니다.

출력 예

```
Verified OK
```

1.5.1. 추가 리소스

- 1.3절. "Tekton 체인에서 데이터에 서명하기 위한 보안"
- 1.2절. "Tekton 체인 구성"

1.6. TEKTON CHAIN을 사용하여 이미지와 검증에 서명 및 확인

클러스터 관리자는 Tekton Chains를 사용하여 다음 작업을 수행하여 image 및 provenances에 서명하고 확인할 수 있습니다.

- 암호화된 x509 키 쌍을 생성하여 Kubernetes 시크릿으로 저장합니다.
- 이미지, 이미지 서명 및 서명된 이미지 테스트를 저장하기 위해 OCI 레지스트리에 대한 인증을 설정합니다.
- 검증 정보를 생성하고 서명하도록 Tekton 체인을 구성합니다.
- 작업 실행에 Kaniko로 이미지를 생성합니다.
- 서명된 이미지와 서명된 인증서를 확인합니다.

사전 요구 사항

다음 사항이 클러스터에 설치되어 있는지 확인합니다.

- Red Hat OpenShift Pipelines Operator
- Tekton 체인
- cosign
- Rekor
- jq

프로세스

1. 암호화된 x509 키 쌍을 생성하여 Kubernetes 시크릿으로 저장합니다.

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

메시지가 표시되면 암호를 입력합니다. cosign은 생성된 개인 키를 **openshift-pipelines** 네임스페이스에 **signing-secrets** Kubernetes 시크릿의 일부로 저장하고 공개 키를 **cosign.pub** 로컬 파일에 씁니다.

2. 이미지 레지스트리에 대한 인증을 구성합니다.

   a. OCI 레지스트리로 서명을 푸시하도록 Tekton 체인 컨트롤러를 구성하려면 작업 실행의 서비스 계정과 연결된 인증 정보를 사용합니다. 자세한 내용은 " OCI 레지스트리에 인증" 섹션을 참조하십시오.

   b. 이미지를 빌드하고 레지스트리로 내보내는 Kaniko 작업에 대한 인증을 구성하려면 필요한 인증 정보가 포함된 docker **config.json** 파일의 Kubernetes 시크릿을 생성합니다.

```
$ oc create secret generic <docker_config_secret_name> \ 1
  --from-file <path_to_config.json> 2
```

   1 docker config secret 이름으로 대체합니다.

   2 docker **config.json** 파일의 경로로 바꿉니다.

3. **chain-config 개체에서 artifacts.taskrun.format,artifacts.taskrun.storage 및 transparency.enabled 매개변수를 설정하여 Tekton 체인을** 구성합니다.

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.format": "in-toto"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.storage": "oci"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"transparency.enabled": "true"}}'
```

4. Kaniko 작업을 시작합니다.

   a. Kaniko 작업을 클러스터에 적용합니다.

```
$ oc apply -f examples/kaniko/kaniko.yaml 1
```

   1 Kaniko 작업의 URI 또는 파일 경로로 바꿉니다.

   b. 적절한 환경 변수를 설정합니다.

```
$ export REGISTRY=<url_of_registry> 1

$ export DOCKERCONFIG_SECRET_NAME=
<name_of_the_secret_in_docker_config_json> 2
```

   1 이미지를 내보낼 레지스트리의 URL을 바꿉니다.

   2 docker **config.json** 파일의 보안 이름으로 바꿉니다.

   c. Kaniko 작업을 시작합니다.

```
$ tkn task start --param IMAGE=$REGISTRY/kaniko-chains --use-param-defaults --
workspace name=source,emptyDir="" --workspace
name=dockerconfig,secret=$DOCKERCONFIG_SECRET_NAME kaniko-chains
```

모든 단계가 완료될 때까지 이 작업의 로그를 관찰합니다. 인증에 성공하면 최종 이미지가 **$REGISTRY/kaniko-chains** 로 푸시됩니다.

5. Tekton **체인에서 인증 정보를 생성하고 서명할 때까지 1분 정도 기다린 다음 작업 실행 시 chain.tekton.dev/signed=true** 주석의 가용성을 확인합니다.

```
$ oc get tr <task_run_name> \ 1
-o json | jq -r .metadata.annotations

{
  "chains.tekton.dev/signed": "true",
  ...
}
```

1 작업 실행 이름으로 대체합니다.

6. 이미지와 인증을 확인합니다.

```
$ cosign verify --key cosign.pub $REGISTRY/kaniko-chains

$ cosign verify-attestation --key cosign.pub $REGISTRY/kaniko-chains
```

7. Rekor에서 이미지의 출처를 확인합니다.

   a. $REGISTRY/kaniko-chains 이미지의 다이제스트를 가져옵니다. 작업 실행을 검색하거나 이미지를 가져와 다이제스트를 추출할 수 있습니다.

   b. Rekor를 검색하여 이미지의 **sha256** 다이제스트와 일치하는 모든 항목을 찾습니다.

```
$ rekor-cli search --sha <image_digest> 1

<uuid_1> 2
<uuid_2> 3
...
```

1 이미지를 **sha256** 다이제스트로 대체합니다.

2 첫 번째 일치는 UUID(Universally unique identifier)입니다.

3 두 번째 일치하는 UUID입니다.

   검색 결과에는 일치하는 항목의 UUID가 표시됩니다. 해당 UUID 중 하나에 인증 정보가 있습니다.

   c. 테스트를 확인하십시오.

```
$ rekor-cli get --uuid <uuid> --format json | jq -r .Attestation | base64 --decode | jq
```

1.7. 추가 리소스

- OpenShift Pipelines 설치

2장. 소프트웨어 공급망 보안 요소를 보려면 웹 콘솔에서 OPENSHIFT PIPELINES 설정

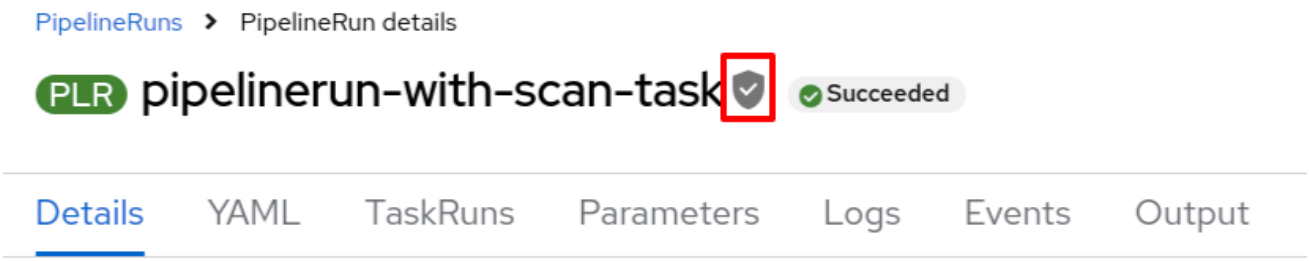
개발자 또는 **관리자** 화면을 사용하여 파이프라인을 생성하거나 수정하고 프로젝트 내의 주요 소프트웨어 공급망 보안 요소를 확인합니다.

다음을 볼 OpenShift Pipelines를 설정합니다.

- **프로젝트 취약점**: 프로젝트 내에서 확인된 취약점을 보여줍니다.
- **Software bill of materials (SBOMs)**: PipelineRun 구성 요소의 자세한 목록을 다운로드하거나 봅니다.

또한 Tekton Chains 요구 사항을 충족하는 PipelineRuns는 이름 옆에 서명된 배지가 표시됩니다. 이 배지는 파이프라인 실행 실행 결과가 암호화 방식으로 서명되어 OCI 이미지와 같이 안전하게 저장됨을 나타냅니다.

그림 2.1. 서명된 배지

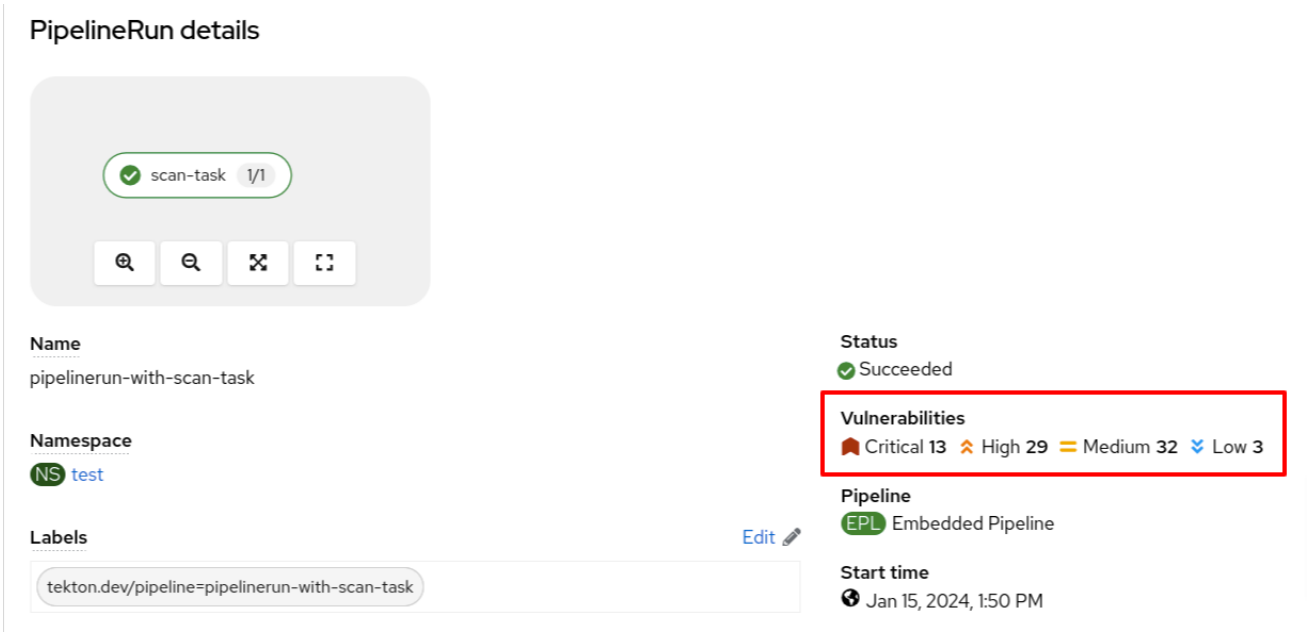


PipelineRun은 Tekton Chain을 구성한 경우에만 해당 이름 옆에 서명된 배지를 표시합니다. Tekton 체인 구성에 대한 자세한 내용은 OpenShift Pipelines 공급망 보안에 Tekton 체인 사용을 참조하십시오.

2.1. 프로젝트 취약점을 보기 위해 OPENSHIFT PIPELINES 설정

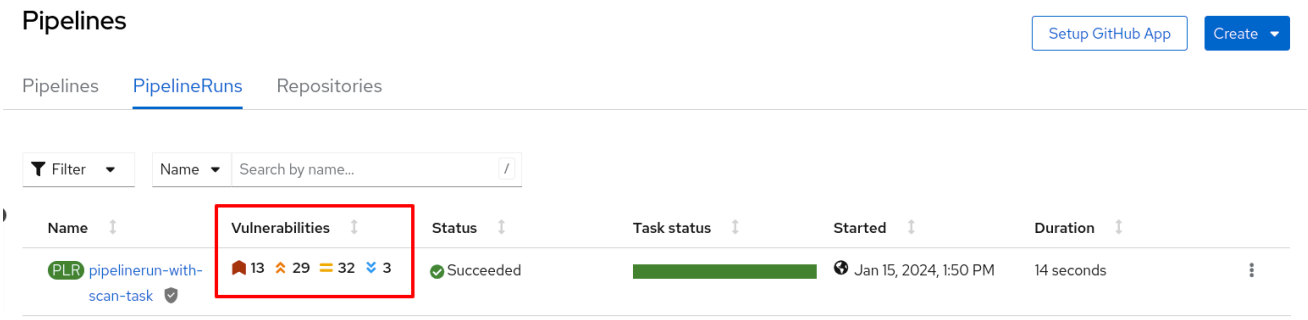
PipelineRun 세부 정보 페이지에는 심각도(심각, 높음, 중간 및 낮음)로 분류된 확인된 취약점이 시각적으로 표시됩니다. 이렇게 간소화된 뷰를 통해 우선 순위 지정 및 수정 작업을 쉽게 수행할 수 있습니다.

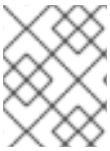
그림 2.2. PipelineRun 세부 정보 페이지의 취약점 보기



파이프라인 실행 목록 보기 페이지의 취약점 열에서 **취약점** 을 검토할 수도 있습니다.

그림 2.3. PipelineRun 목록 보기의 취약점 보기





참고

확인된 취약점의 시각적 표현은 OpenShift Container Platform 버전 4.15 릴리스에서 사용할 수 있습니다.

사전 요구 사항

- 웹 콘솔에 로그인했습니다.
- 프로젝트에 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성할 적절한 역할과 권한이 있습니다.
- 기존 취약점 검사 작업이 있습니다.

프로시저

1. **개발자** 또는 **관리자** 관점에서 취약점을 시각적으로 표시하려는 관련 프로젝트로 전환합니다.
2. 기존 취약점 검사 작업을 업데이트하여 출력을 .json 파일에 저장한 다음 다음 형식으로 취약점 요약을 추출합니다.

```
# The format to extract vulnerability summary (adjust the jq command for different JSON structures).
jq -rce \
   '{vulnerabilities:{
     critical: (.result.summary.CRITICAL),
     high: (.result.summary.IMPORTANT),
     medium: (.result.summary.MODERATE),
     low: (.result.summary.LOW)
     }}' scan_output.json | tee $(results.SCAN_OUTPUT.path)
```

참고

다른 JSON 구조에 대해 jq 명령을 조정해야 할 수도 있습니다.

a. (선택 사항) 취약점 검사 작업이 없는 경우 다음 형식으로 생성합니다.
Roxctl을 사용한 취약점 검사 작업의 예

```
apiVersion: tekton.dev/v1
```

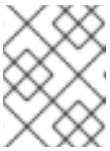
```
kind: Task
metadata:
  name: vulnerability-scan 1
  annotations:
    task.output.location: results 2
    task.results.format: application/json
    task.results.key: SCAN_OUTPUT 3
spec:
  results:
    - description: CVE result format 4
      name: SCAN_OUTPUT
  steps:
    - name: roxctl 5
      image: quay.io/roxctl-tool-image 6
      env:
        - name: ENV_VAR_NAME_1 7
          valueFrom:
            secretKeyRef:
              key: secret_key_1
              name: secret_name_1
      env:
        - name: ENV_VAR_NAME_2
          valueFrom:
            secretKeyRef:
              key: secret_key_2
              name: secret_name_2
      script: | 8
        #!/bin/sh

        # Sample shell script

        echo "ENV_VAR_NAME_1: " $ENV_VAR_NAME_1
        echo "ENV_VAR_NAME_2: " $ENV_VAR_NAME_2
        jq --version (adjust the jq command for different JSON structures)
        curl -k -L -H "Authorization: Bearer $ENV_VAR_NAME_1"
https://$ENV_VAR_NAME_2/api/cli/download/roxctl-linux --output ./roxctl
        chmod +x ./roxctl
        echo "roxctl version"
        ./roxctl version
        echo "image from pipeline: "

        # Replace the following line with your dynamic image logic
        DYNAMIC_IMAGE=$(get_dynamic_image_logic_here)
        echo "Dynamic image: $DYNAMIC_IMAGE"
        ./roxctl image scan --insecure-skip-tls-verify -e $ENV_VAR_NAME_2 --image
$DYNAMIC_IMAGE --output json  > roxctl_output.json
        more roxctl_output.json
        jq -rce \ 9
          '{vulnerabilities:{
          critical: (.result.summary.CRITICAL),
          high: (.result.summary.IMPORTANT),
          medium: (.result.summary.MODERATE),
          low: (.result.summary.LOW)
            }}' scan_output.json | tee $(results.SCAN_OUTPUT.path)
```

1. 작업 이름입니다.
2. 작업 출력을 저장하는 위치입니다.
3. 검사 작업 결과의 이름 지정 규칙입니다. 유효한 명명 규칙은 **SCAN_OUTPUT** 문자열로 끝나야 합니다. 예를 들어 SCAN_OUTPUT, MY_CUSTOM_SCAN_OUTPUT 또는 ACS_SCAN_OUTPUT.
4. 결과에 대한 설명입니다.
5. 사용한 취약점 검사 도구의 이름입니다.
6. 검사 도구가 포함된 실제 이미지의 위치입니다.
7. 툴별 환경 변수입니다.
8. json 출력으로 실행할 쉘 스크립트입니다. 예를 들면 scan_output.json입니다.
9. 취약점 요약을 추출하는 형식(다른 JSON 구조에 대해 **jq** 명령 조정).



참고

이는 구성의 예입니다. 특정 검사 도구에 따라 값을 수정하여 예상 형식으로 결과를 설정합니다.

3. 적절한 Pipeline을 업데이트하여 다음 형식으로 취약점 사양을 추가합니다.

```
...
spec:
  results:
    - description: The common vulnerabilities and exposures (CVE) result
      name: SCAN_OUTPUT
      value: $(tasks.vulnerability-scan.results.SCAN_OUTPUT)
```

검증

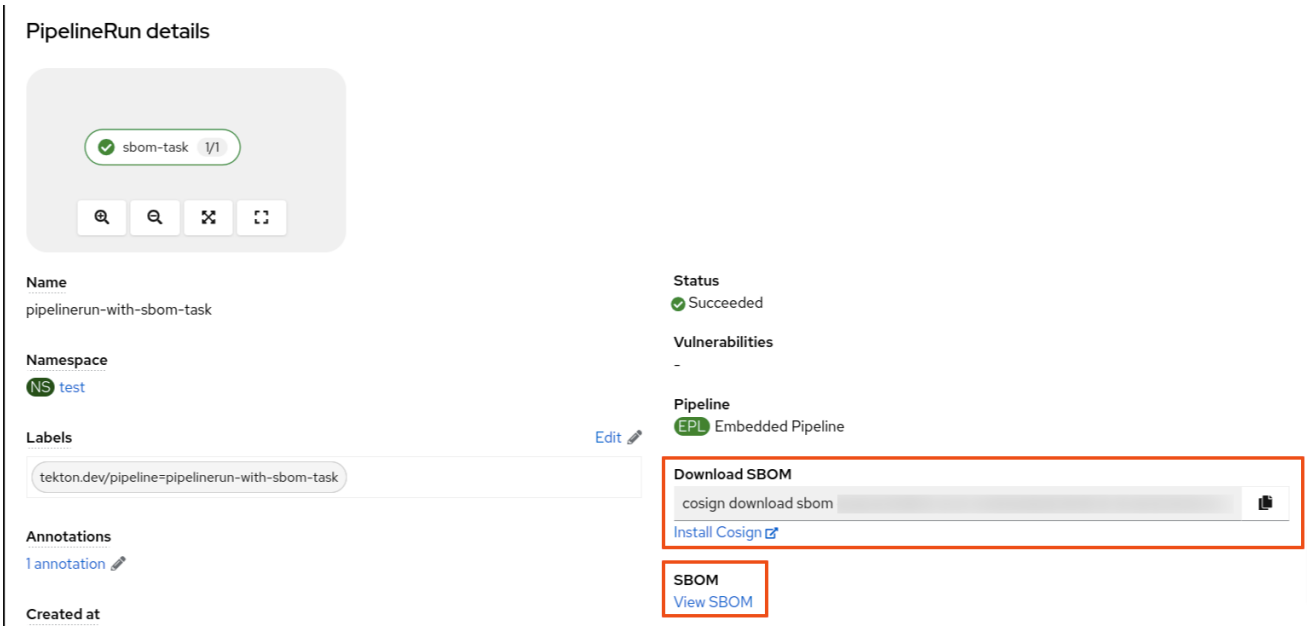
- **PipelineRun** 세부 정보 페이지로 이동하여 확인된 **취약점** 의 시각적 표현에 대한 취약점 행을 검토합니다.
- 또는 **PipelineRun** 목록 보기 페이지로 이동하여 **Vulnerabilities** 열을 검토할 수 있습니다.

2.2. SBOM을 다운로드하거나 볼 수 있도록 OPENSHIFT PIPELINES 설정

PipelineRun 세부 정보 페이지에서는 SBOM(Software bill of materials)을 다운로드하거나 볼 수 있는 옵션을 제공하여 공급망 내 투명성과 제어를 강화할 수 있습니다. SBOM은 구성 요소에서 사용하는 모든 소프트웨어 라이브러리를 나열합니다. 이러한 라이브러리는 특정 기능을 활성화하거나 개발을 용이하게 할 수 있습니다.

SBOM을 사용하여 소프트웨어의 구성을 더 잘 이해하고 취약점을 식별하고 발생할 수 있는 보안 문제의 잠재적 영향을 평가할 수 있습니다.

그림 2.4. SBOM을 다운로드하거나 볼 수 있는 옵션



사전 요구 사항

- 웹 콘솔에 로그인했습니다.
- 프로젝트에 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성할 적절한 역할과 권한이 있습니다.

프로세스

1. **개발자** 또는 **관리자** 관점에서 SBOM의 시각적 표현이 필요한 관련 프로젝트로 전환합니다.
2. 다음 형식으로 작업을 추가하여 SBOM 정보를 보거나 다운로드합니다.

   SBOM 작업 예

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: sbom-task (1)
  annotations:
    task.output.location: results (2)
    task.results.format: application/text
    task.results.key: LINK_TO_SBOM (3)
    task.results.type: external-link (4)
spec:
  results:
    - description: Contains the SBOM link (5)
      name: LINK_TO_SBOM
  steps:
    - name: print-sbom-results
      image: quay.io/image (6)
      script: | (7)
        #!/bin/sh
        syft version
        syft quay.io/<username>/quarkus-demo:v2 --output cyclonedx-json=sbom-image.json
```

```
      echo 'BEGIN SBOM'
      cat sbom-image.json
      echo 'END SBOM'
      echo 'quay.io/user/workloads/<namespace>/node-express/node-express:build-8e536-
1692702836' | tee $(results.LINK_TO_SBOM.path) 8
```

1. 작업 이름입니다.
2. 작업 출력을 저장하는 위치입니다.
3. SBOM 작업 결과 이름입니다. SBOM 결과 작업의 이름을 변경하지 마십시오.
4. (선택 사항) 새 탭에서 SBOM을 열도록 설정합니다.
5. 결과에 대한 설명입니다.
6. SBOM을 생성하는 이미지입니다.
7. SBOM 이미지를 생성하는 스크립트입니다.
8. 경로 이름과 SBOM 이미지입니다.

3. 새로 생성된 SBOM 작업을 참조하도록 Pipeline을 업데이트합니다.

```
...
spec:
  tasks:
    - name: sbom-task
      taskRef:
        name: sbom-task 1
  results:
    - name: IMAGE_URL 2
      description: url
      value: <oci_image_registry_url> 3
```

1. 2단계에서 만든 것과 동일한 이름입니다.
2. 결과 이름입니다.
3. **.sbom** 이미지를 포함하는 OCI 이미지 리포지토리 URL입니다.

4. 영향을 받는 OpenShift 파이프라인을 다시 실행합니다.

2.2.1. 웹 UI에서 SBOM 보기

사전 요구 사항

- SBOM을 다운로드하거나 볼 수 있도록 OpenShift Pipelines를 설정해야 합니다.

프로세스

1. 활동 → PipelineRuns 탭으로 이동합니다.

2. 보려는 SBOM이 있는 프로젝트의 경우 최신 파이프라인 실행을 선택합니다.
3. **PipelineRun** 세부 정보 페이지에서 **SBOM 보기**를 선택합니다.
   a. 웹 브라우저를 사용하여 SBOM에서 소프트웨어 공급망의 취약점을 나타내는 용어를 즉시 검색할 수 있습니다. 예를 들어 **log4j** 를 검색해 보십시오.
   b. **다운로드**를 선택하여 SBOM을 다운로드하거나 **Expand** 를 선택하여 전체 화면을 볼 수 있습니다.

2.2.2. CLI에서 SBOM 다운로드

사전 요구 사항

- Cosign CLI 툴을 설치했습니다.
- SBOM을 다운로드하거나 볼 수 있도록 OpenShift Pipelines를 설정해야 합니다.

프로세스

1. 터미널을 열고 **개발자** 또는 **관리자** 화면에 로그인한 다음 관련 프로젝트로 전환합니다.
2. OpenShift 웹 콘솔에서 **다운로드 sbom** 명령을 복사하여 터미널에서 실행합니다.

   cosign 명령 예

   ```
   $ cosign download sbom quay.io/<workspace>/user-workload@sha256
   ```

   a. (선택 사항) 검색 가능한 형식으로 전체 SBOM을 보려면 다음 명령을 실행하여 출력을 리디렉션합니다.

      cosign 명령 예

      ```
      $ cosign download sbom quay.io/<workspace>/user-workload@sha256 > sbom.txt
      ```

2.2.3. SBOM 읽기

SBOM에서 다음 샘플 발췌 내용에서 볼 수 있듯이 프로젝트에서 사용하는 각 라이브러리의 4가지 특성을 확인할 수 있습니다.

- 작성자 또는 게시자
- 이름
- 해당 버전
- 라이센스

이 정보는 개별 라이브러리가 안전하게 소싱, 업데이트 및 준수하는지 확인하는 데 도움이 됩니다.

SBOM 예

```
{
  "bomFormat": "CycloneDX",
```

```
"specVersion": "1.4",
"serialNumber": "urn:uuid:89146fc4-342f-496b-9cc9-07a6a1554220",
"version": 1,
"metadata": {
    ...
},
"components": [
    {
        "bom-ref": "pkg:pypi/flask@2.1.0?package-id=d6ad7ed5aac04a8",
        "type": "library",
        "author": "Armin Ronacher <armin.ronacher@active-4.com>",
        "name": "Flask",
        "version": "2.1.0",
        "licenses": [
            {
                "license": {
                    "id": "BSD-3-Clause"
                }
            }
        ],
        "cpe": "cpe:2.3:a:armin-ronacher:python-Flask:2.1.0:*:*:*:*:*:*:*",
        "purl": "pkg:pypi/Flask@2.1.0",
        "properties": [
            {
                "name": "syft:package:foundBy",
                "value": "python-package-cataloger"
                ...
```


2.3. 추가 리소스

- 웹 콘솔에서 Red Hat OpenShift Pipelines 작업

3장. POD의 보안 컨텍스트 구성

OpenShift Pipelines가 시작되는 Pod의 기본 서비스 계정은 **파이프라인** 입니다. **파이프라인** 서비스 계정과 연결된 SCC(보안 컨텍스트 제약 조건)는 **pipelines-scc** 입니다. **pipelines-scc** SCC는 다음 YAML 사양에 정의된 대로 약간의 차이가 있는 **anyuid** SCC를 기반으로 합니다.

pipelines-scc.yaml 스니펫 예

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
# ...
allowedCapabilities:
  - SETFCAP
# ...
fsGroup:
  type: MustRunAs
# ...
```

또한 OpenShift Pipelines의 일부로 제공되는 **Buildah** 클러스터 작업은 **vfs** 를 기본 스토리지 드라이버로 사용합니다.

OpenShift Pipelines가 파이프라인 실행 및 작업 실행을 위해 생성하는 Pod의 보안 컨텍스트를 구성할 수 있습니다. 다음과 같은 변경을 수행할 수 있습니다.

- 모든 Pod의 기본 SCC 및 최대 SCC 변경
- 특정 네임스페이스에서 파이프라인 실행 및 작업 실행을 위해 생성된 Pod의 기본 SCC 변경
- 사용자 정의 SCC 및 서비스 계정을 사용하도록 특정 파이프라인 실행 또는 작업 실행 구성

참고

buildah 를 실행하여 모든 이미지를 빌드할 수 있도록 하는 가장 간단한 방법은 **권한이 있는** SCC를 사용하여 Pod에서 root로 실행하는 것입니다. 보다 제한적인 보안 설정으로 **buildah** 를 실행하는 방법에 대한 자세한 내용은 루트가 아닌 사용자로 Buildah를 사용하여 컨테이너 이미지 빌드 를 참조하십시오.

3.1. OPENSHIFT PIPELINES에서 생성하는 POD의 기본 및 최대 SCC 구성

OpenShift Pipelines가 작업 실행 및 파이프라인 실행을 위해 생성하는 모든 Pod에 대해 기본 SCC(보안 컨텍스트 제약 조건)를 구성할 수 있습니다. 모든 네임스페이스에서 이러한 Pod에 대해 구성할 수 있는 최소 제한적인 SCC인 최대 SCC를 구성할 수도 있습니다.

프로세스

- 다음 명령을 입력하여 **TektonConfig** CR(사용자 정의 리소스)을 편집합니다.

```
$ oc edit TektonConfig config
```

다음 예와 같이 사양에서 default 및 maximum SCC를 설정합니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
```

```
  name: config
spec:
# ...
  platforms:
    openshift:
      scc:
        default: "restricted-v2" (1)
        maxAllowed: "privileged" (2)
```

(1) **spec.platforms.openshift.scc.default** 는 기본적으로 **파이프라인** SA인 워크로드에 사용되는 서비스 계정(SA)에 OpenShift Pipelines를 연결하는 기본 SCC를 지정합니다. 이 SCC는 모든 파이프라인 실행 및 작업 실행 Pod에 사용됩니다.

(2) **spec.platforms.openshift.scc.maxAllowed** 는 네임스페이스에서 파이프라인 실행 및 작업 실행 Pod에 대해 구성할 수 있는 가장 덜 제한적인 SCC를 지정합니다. 이 설정은 특정 파이프라인 실행 또는 작업 실행에서 사용자 지정 SA 및 SCC를 구성할 때 적용되지 않습니다.

추가 리소스

- OpenShift Pipelines의 기본 서비스 계정 변경

3.2. 네임스페이스에서 POD의 SCC 구성

OpenShift Pipelines가 특정 네임스페이스에서 생성하는 파이프라인 실행 및 작업 실행에 대해 OpenShift Pipelines가 생성하는 모든 Pod에 대해 SCC(보안 컨텍스트 제약 조건)를 구성할 수 있습니다. 이 SCC는 **spec.platforms.openshift.scc.maxAllowed** 사양에서 **TektonConfig** CR을 사용하여 구성한 최대 SCC보다 덜 제한적이지 않아야 합니다.

프로세스

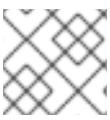
- 네임스페이스의 **operator.tekton.dev/scc** 주석을 SCC 이름으로 설정합니다.

  OpenShift Pipelines Pod에 대한 **SCC**를 구성하기 위한 **namespace** 주석의 예

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  annotations:
    operator.tekton.dev/scc: nonroot
```

3.3. 사용자 정의 SCC 및 사용자 정의 서비스 계정을 사용하여 파이프라인 실행 및 작업 실행

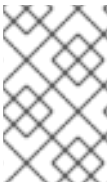
기본 파이프라인 서비스 계정과 연결된 **pipelines -scc** SCC(보안 컨텍스트 제약 조건)를 사용하는 경우 파이프라인 실행 및 작업 실행 Pod가 시간 초과에 직면할 수 있습니다. 이는 기본 **pipelines-scc** SCC에서 **fsGroup.type** 매개변수가 **MustRunAs** 로 설정되어 있기 때문에 발생합니다.



참고

Pod 시간 초과에 대한 자세한 내용은 BZ#1995779 를 참조하십시오.

Pod 시간 초과를 방지하려면 **fsGroup.type** 매개변수를 **RunAsAny** 로 설정하여 사용자 지정 SCC를 생성하고 사용자 지정 서비스 계정과 연결할 수 있습니다.



참고

가장 좋은 방법은 파이프라인 실행 및 작업 실행을 위해 사용자 정의 SCC 및 사용자 정의 서비스 계정을 사용합니다. 이 접근 방식을 사용하면 유연성이 향상되고 업그레이드 중에 기본값이 수정될 때 실행이 중단되지 않습니다.

프로세스

1. **fsGroup.type** 매개변수를 **RunAsAny** 로 설정하여 사용자 지정 SCC를 정의합니다.

   예: 사용자 정의 **SCC**

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: my-scc is a close replica of anyuid scc. pipelines-scc has
fsGroup - RunAsAny.
  name: my-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

2. 사용자 지정 SCC를 생성합니다.

   예: **my-scc SCC** 생성

예: **my-scc SCC** 생성

```
$ oc create -f my-scc.yaml
```

3. 사용자 정의 서비스 계정을 생성합니다.

   예: **fsgroup-runasany** 서비스 계정 생성

   ```
   $ oc create serviceaccount fsgroup-runasany
   ```

4. 사용자 지정 SCC를 사용자 지정 서비스 계정과 연결합니다.

   예: **my-scc SCC**를 **fsgroup-runasany** 서비스 계정과 연결

   ```
   $ oc adm policy add-scc-to-user my-scc -z fsgroup-runasany
   ```

   권한 있는 작업에 사용자 정의 서비스 계정을 사용하려면 다음 명령을 실행하여 **권한 있는** SCC를 사용자 정의 서비스 계정과 연결할 수 있습니다.

   예: **권한 있는 SCC**를 **fsgroup-runasany** 서비스 계정에 연결

   ```
   $ oc adm policy add-scc-to-user privileged -z fsgroup-runasany
   ```

5. 파이프라인 실행 및 작업 실행에서 사용자 정의 서비스 계정을 사용합니다.

   예: 파이프라인은 **fsgroup-runasany** 사용자 지정 서비스 계정으로 **YAML**을 실행합니다.

   ```
   apiVersion: tekton.dev/v1
   kind: PipelineRun
   metadata:
     name: <pipeline-run-name>
   spec:
     pipelineRef:
       name: <pipeline-cluster-task-name>
     taskRunTemplate:
       serviceAccountName: 'fsgroup-runasany'
   ```

   예: 작업에서는 **fsgroup-runasany** 사용자 지정 서비스 계정으로 **YAML**을 실행합니다.

   ```
   apiVersion: tekton.dev/v1
   kind: TaskRun
   metadata:
     name: <task-run-name>
   spec:
     taskRef:
       name: <cluster-task-name>
     taskRunTemplate:
       serviceAccountName: 'fsgroup-runasany'
   ```

3.4. 추가 리소스

- 보안 컨텍스트 제약 조건 관리

4장. 이벤트 리스너로 WEBHOOK 보안

관리자는 이벤트 리스너를 사용하여 Webhook를 보호할 수 있습니다. 네임스페이스를 생성한 후 네임스페이스에 **operator.tekton.dev/enable-annotation=enabled** 레이블을 추가하여 **Eventlistener** 리소스의 HTTPS를 활성화합니다. 그런 다음 재암호화 TLS 종료를 사용하여 **Trigger** 리소스 및 보안 경로를 생성합니다.

Red Hat OpenShift Pipelines에서 트리거는 **Eventlistener** 리소스에 대한 비보안 HTTP 및 보안 HTTPS 연결을 모두 지원합니다. HTTPS는 클러스터 내부 및 외부의 연결을 보호합니다.

Red Hat OpenShift Pipelines는 네임스페이스의 레이블을 감시하는 **tekton-operator-proxy-webhook** Pod를 실행합니다. 네임스페이스에 라벨을 추가하면 Webhook에서 **EventListener** 오브젝트에 **service.beta.openshift.io/serving-cert-secret-name=<secret_name>** 주석을 설정합니다. 이를 통해 시크릿과 필수 인증서를 생성합니다.

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

또한 생성된 시크릿을 **Eventlistener** pod 마운트하여 요청을 보호할 수 있습니다.

4.1. OPENSHIFT 경로와의 보안 연결 제공

재암호화 TLS 종료로 경로를 생성합니다.

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=<hostname>
```

또는 재암호화된 TLS 종료 YAML 파일을 만들어 보안 경로를 만들 수도 있습니다.

보안 경로를 생성하기 위해 **TLS** 종료 **YAML**을 재암호화하는 예

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend (2)
  tls:
    termination: reencrypt (3)
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- (4)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1) (2) 63자로 제한되는 개체의 이름입니다.

(3) 종료 필드는 **reencrypt**로 설정됩니다. 이 필드는 유일한 필수 TLS 필드입니다.

4 이는 재암호화에 필요합니다. **destinationCACertificate** 필드는 엔드포인트 인증서의 유효성을 검사하고 라우터에서 대상 pod로의 연결을 보호합니다. 다음 시나리오 중 하나에서 이 필드를 생략할 수

- 서비스는 서비스 서명 인증서를 사용합니다.
- 관리자는 라우터의 기본 CA 인증서를 지정하고 서비스에는 해당 CA에서 서명한 인증서가 있습니다.

oc create route reencrypt --help 명령을 실행하여 더 많은 옵션을 표시할 수 있습니다.

4.2. 보안 HTTPS 연결을 사용하여 샘플 EVENTLISTENER 리소스 생성

이 섹션에서는 pipelines-tutorial 예제를 사용하여 보안 HTTPS 연결을 사용하여 샘플 EventListener 리소스 생성을 만드는 방법을 보여줍니다.

프로세스

1. pipelines-tutorial 리포지토리에서 사용 가능한 YAML 파일에서 **TriggerBinding** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/01_binding.yaml
```

2. pipelines-tutorial 리포지토리에서 직접 **TriggerTemplate** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/02_template.yaml
```

3. pipelines-tutorial 리포지토리에서 직접 **Trigger** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/03_trigger.yaml
```

4. 보안 HTTPS 연결을 사용하여 **EventListener** 리소스를 생성합니다.
   a. **Eventlistener** 리소스에 대한 보안 HTTPS 연결을 활성화하려면 레이블을 추가합니다.

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

   b. pipelines-tutorial 리포지토리에서 사용 가능한 YAML 파일에서 **EventListener** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/04_event_listener.yaml
```

   c. 재암호화 TLS 종료로 경로를 생성합니다.

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

5장. 보안을 사용하여 리포지터리로 파이프라인 인증

파이프라인 및 작업에는 Git 리포지토리 및 컨테이너 리포지토리로 인증하기 위해 인증 정보가 필요할 수 있습니다. Red Hat OpenShift Pipelines에서는 시크릿을 사용하여 실행 중에 Git 리포지토리 또는 컨테이너 리포지토리와 상호 작용하는 파이프라인 실행 및 작업 실행을 인증할 수 있습니다.

Git 리포지토리를 사용한 인증에 대한 *시크릿을 Git 시크릿* 이라고 합니다.

파이프라인 실행 또는 작업 실행에서는 연결된 서비스 계정을 통해 시크릿에 대한 액세스 권한을 얻습니다. 또는 파이프라인 또는 작업에서 작업 공간을 정의하고 해당 시크릿을 작업 공간에 바인딩할 수 있습니다.

5.1. 사전 요구 사항

- **oc** OpenShift 명령줄 유틸리티를 설치했습니다.

5.2. 서비스 계정을 사용하여 보안 제공

서비스 계정을 사용하여 Git 리포지토리 및 컨테이너 리포지토리를 통한 인증에 대한 시크릿을 제공할 수 있습니다.

시크릿을 서비스 계정과 연결할 수 있습니다. 시크릿의 정보는 이 서비스 계정에서 실행되는 작업에 사용할 수 있습니다.

5.2.1. 서비스 계정의 보안 유형 및 주석

서비스 계정을 사용하여 인증 보안을 제공하는 경우 OpenShift Pipelines는 여러 가지 보안 유형을 지원합니다. 이러한 보안 유형의 대부분은 인증 보안이 유효한 리포지토리를 정의하는 주석을 제공해야 합니다.

5.2.1.1. Git 인증 보안

서비스 계정을 사용하여 인증 보안을 제공하는 경우 OpenShift Pipelines는 Git 인증에 대해 다음 유형의 보안을 지원합니다.

- **kubernetes.io/basic-auth**: 기본 인증을 위한 사용자 이름과 암호
- **kubernetes.io/ssh-auth**: SSH 기반 인증을 위한 키

서비스 계정을 사용하여 인증 보안을 제공하는 경우 Git 시크릿에 하나 이상의 주석 키가 있어야 합니다. 각 키의 이름은 **tekton.dev/git-** 로 시작해야 하며 값은 OpenShift Pipelines에서 시크릿의 인증 정보를 사용해야 하는 호스트의 URL입니다.

다음 예에서 OpenShift Pipelines는 **basic-auth** 시크릿을 사용하여 **github.com** 및 **gitlab.com** 의 리포지토리에 액세스합니다.

예: 여러 Git 리포지토리를 사용한 기본 인증을 위한 인증 정보

```
apiVersion: v1
kind: Secret
metadata:
  name: git-secret-basic
  annotations:
    tekton.dev/git-0: github.com
    tekton.dev/git-1: gitlab.com
```

```
type: kubernetes.io/basic-auth
stringData:
  username: <username> 1
  password: <password> 2
```

1 리포지토리의 사용자 이름

2 리포지토리의 암호 또는 개인 액세스 토큰

다음 예와 같이 **ssh-auth** 시크릿을 사용하여 Git 리포지토리에 액세스하기 위한 개인 키를 제공할 수도 있습니다.

예: SSH 기반 인증을 위한 개인 키

```
apiVersion: v1
kind: Secret
metadata:
  name: git-secret-ssh
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 1
```

1 SSH 개인 키 파일의 내용입니다.

5.2.1.2. 컨테이너 레지스트리 인증 보안

서비스 계정을 사용하여 인증 보안을 제공하는 경우 OpenShift Pipelines는 컨테이너(Docker) 레지스트리 인증에 대해 다음 유형의 보안을 지원합니다.

- **kubernetes.io/basic-auth**: 기본 인증을 위한 사용자 이름과 암호
- **kubernetes.io/dockercfg**: 직렬화된 **~/.dockercfg** 파일
- **kubernetes.io/dockerconfigjson**: 직렬화된 **~/.docker/config.json** 파일

서비스 계정을 사용하여 인증 보안을 제공하는 경우 **kubernetes.io/basic-auth** 유형의 컨테이너 레지스트리 시크릿에 하나 이상의 주석 키가 있어야 합니다. 각 키의 이름은 **tekton.dev/docker-** 로 시작해야 하며 값은 OpenShift Pipelines에서 시크릿의 인증 정보를 사용해야 하는 호스트의 URL입니다. 이 주석은 다른 유형의 컨테이너 레지스트리 시크릿에는 필요하지 않습니다.

다음 예에서 OpenShift Pipelines는 사용자 이름과 암호를 사용하는 **basic-auth** 시크릿을 사용하여 **quay.io** 및 **my-registry.example.com** 에서 컨테이너 레지스트리에 액세스합니다.

예: 여러 컨테이너 리포지토리를 사용한 기본 인증을 위한 인증 정보

```
apiVersion: v1
kind: Secret
metadata:
  name: docker-secret-basic
  annotations:
    tekton.dev/docker-0: quay.io
```

```
    tekton.dev/docker-1: my-registry.example.com
type: kubernetes.io/basic-auth
stringData:
  username: <username> 1
  password: <password> 2
```

1 레지스트리의 사용자 이름

2 레지스트리의 암호 또는 개인 액세스 토큰

다음 예와 같이 기존 구성 파일에서 **kubernetes.io/dockercfg** 및 **kubernetes.io/dockerconfigjson** 시크릿을 생성할 수 있습니다.

예: 기존 구성 파일에서 컨테이너 리포지토리에 인증하기 위한 시크릿을 생성하는 명령

```
$ oc create secret generic docker-secret-config \
    --from-file=config.json=/home/user/.docker/config.json \
    --type=kubernetes.io/dockerconfigjson
```

다음 예제와 같이 **oc** 명령줄 유틸리티를 사용하여 인증 정보에서 **kubernetes.io/dockerconfigjson** 시크릿을 생성할 수도 있습니다.

예: 인증 정보에서 컨테이너 리포지토리에 인증하기 위한 시크릿을 생성하는 명령

```
$ oc create secret docker-registry docker-secret-config \
  --docker-email=<email> \ 1
  --docker-username=<username> \ 2
  --docker-password=<password> \ 3
  --docker-server=my-registry.example.com:5000 4
```

1 레지스트리의 이메일 주소

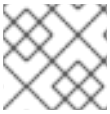
2 레지스트리의 사용자 이름

3 레지스트리의 암호 또는 개인 액세스 토큰

4 레지스트리의 호스트 이름 및 포트

5.2.2. 서비스 계정을 사용하여 Git에 대한 기본 인증 구성

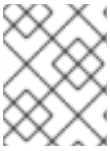
암호 보호 리포지토리에서 리소스를 검색하는 파이프라인의 경우 해당 파이프라인에 대한 기본 인증을 구성할 수 있습니다.



참고

기본 인증이 아닌 SSH 기반 인증을 사용하는 것이 좋습니다.

파이프라인에 대한 기본 인증을 구성하려면 기본 인증 보안을 생성하고 이 보안을 서비스 계정과 연결한 다음, 이 서비스 계정을 **TaskRun** 또는 **PipelineRun** 리소스와 연결합니다.



참고

GitHub의 경우 일반 암호를 사용한 인증은 더 이상 사용되지 않습니다. 대신 개인 액세스 토큰 을 사용합니다.

프로세스

1. **secret.yaml** 파일에서 시크릿의 YAML 매니페스트를 생성합니다. 이 매니페스트에서 사용자 이름과 암호 또는 GitHub 개인 액세스 토큰 을 지정하여 대상 Git 리포지토리에 액세스합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: basic-user-pass (1)
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/basic-auth
stringData:
  username: <username> (2)
  password: <password> (3)
```

(1) 보안의 이름입니다. 예에서는 **basic-user-pass** 입니다.

(2) Git 리포지토리의 사용자 이름입니다.

(3) Git 리포지토리의 암호 또는 개인 액세스 토큰입니다.

2. **serviceaccount.yaml** 파일에서 서비스 계정에 대한 YAML 매니페스트를 생성합니다. 이 매니페스트에서 보안을 서비스 계정과 연결합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot (1)
secrets:
  - name: basic-user-pass (2)
```

(1) 서비스 계정의 이름입니다. 이 예에서는 **build-bot** 입니다.

(2) 보안의 이름입니다. 예에서는 **basic-user-pass** 입니다.

3. **run.yaml** 파일에서 작업 실행 또는 파이프라인 실행에 대한 YAML 매니페스트를 생성하고 서비스 계정을 작업 실행 또는 파이프라인 실행과 연결합니다. 다음 예제 중 하나를 사용합니다.

- 서비스 계정을 **TaskRun** 리소스와 연결합니다.

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: build-push-task-run-2 (1)
spec:
  taskRunTemplate:
```

```
    serviceAccountName: build-bot (2)
  taskRef:
    name: build-push (3)
```

(1) 작업 실행의 이름입니다. 예에서는 **build-push-task-run-2** 입니다.

(2) 서비스 계정의 이름입니다. 이 예에서는 **build-bot** 입니다.

(3) 작업 이름입니다. 예에서는 **build-push**.

- 서비스 계정을 **PipelineRun** 리소스와 연결합니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: demo-pipeline (1)
  namespace: default
spec:
  taskRunTemplate:
    serviceAccountName: build-bot (2)
  pipelineRef:
    name: demo-pipeline (3)
```

(1) 파이프라인 실행의 이름입니다. 예에서는 **demo-pipeline** 입니다.

(2) 서비스 계정의 이름입니다. 이 예에서는 **build-bot** 입니다.

(3) 파이프라인의 이름입니다. 예에서는 **demo-pipeline** 입니다.

4. 다음 명령을 입력하여 생성한 YAML 매니페스트를 적용합니다.

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

5.2.3. 서비스 계정을 사용하여 Git에 대한 SSH 인증 구성

SSH 키를 사용하여 구성된 리포지토리에서 리소스를 검색하려면 파이프라인에 대한 SSH 기반 인증을 구성해야 합니다.

파이프라인에 대한 SSH 기반 인증을 구성하려면 SSH 개인 키를 사용하여 인증 시크릿을 생성하고 이 보안을 서비스 계정과 연결한 다음, 이 서비스 계정을 **TaskRun** 또는 **PipelineRun** 리소스와 연결합니다.

프로세스

1. SSH 개인 키를 생성하거나 일반적으로 **~/.ssh/id_rsa** 파일에서 사용할 수 있는 기존 개인 키를 복사합니다.

2. **secret.yaml** 파일에서 시크릿의 YAML 매니페스트를 생성합니다. 이 매니페스트에서 **ssh-privatekey** 의 값을 SSH 개인 키 파일의 콘텐츠로 설정하고 **known_hosts** 값을 알려진 호스트 파일의 콘텐츠로 설정합니다.

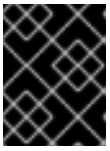
```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: ssh-key 1
  annotations:
    tekton.dev/git-0: github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 2
  known_hosts: 3
```

1 SSH 개인 키가 포함된 시크릿의 이름입니다. 예에서는 **ssh-key**.

2 SSH 개인 키 파일의 내용입니다.

3 알려진 호스트 파일의 내용입니다.



중요

알려진 호스트 파일을 생략하면 OpenShift Pipelines에서 서버의 공개 키를 수락합니다.

3. 선택 사항: 주석 값 끝에 **:<port_number** >를 추가하여 사용자 정의 SSH 포트를 지정합니다. 예: **tekton.dev/git-0: github.com:2222**.

4. **serviceaccount.yaml** 파일에서 서비스 계정에 대한 YAML 매니페스트를 생성합니다. 이 매니페스트에서 보안을 서비스 계정과 연결합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot 1
secrets:
  - name: ssh-key 2
```

1 서비스 계정의 이름입니다. 이 예에서는 **build-bot** 입니다.

2 SSH 개인 키가 포함된 시크릿의 이름입니다. 예에서는 **ssh-key**.

5. **run.yaml** 파일에서 서비스 계정을 작업 실행 또는 파이프라인 실행과 연결합니다. 다음 예제 중 하나를 사용합니다.

   - 서비스 계정을 작업과 연결하려면 다음을 실행합니다.

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: build-push-task-run-2 1
spec:
  taskRunTemplate:
    serviceAccountName: build-bot 2
  taskRef:
    name: build-push 3
```

1 작업 실행의 이름입니다. 예에서는 **build-push-task-run-2** 입니다.

2 서비스 계정의 이름입니다. 이 예에서는 **build-bot** 입니다.

3 작업 이름입니다. 예에서는 **build-push**.

- 서비스 계정을 파이프라인과 연결하려면 다음을 실행합니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: demo-pipeline 1
  namespace: default
spec:
  taskRunTemplate:
    serviceAccountName: build-bot 2
  pipelineRef:
    name: demo-pipeline 3
```

1 파이프라인 실행의 이름입니다. 예에서는 **demo-pipeline** 입니다.

2 서비스 계정의 이름입니다. 이 예에서는 **build-bot** 입니다.

3 파이프라인의 이름입니다. 예에서는 **demo-pipeline** 입니다.

6. 변경 사항을 적용합니다.

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

5.2.4. 서비스 계정을 사용하여 컨테이너 레지스트리 인증 구성

레지스트리에서 컨테이너 이미지를 검색하거나 컨테이너 이미지를 레지스트리로 푸시하려면 파이프라인의 경우 해당 레지스트리에 대한 인증을 구성해야 합니다.

파이프라인에 대한 레지스트리 인증을 구성하려면 Docker 구성 파일을 사용하여 인증 보안을 생성하고 이 보안을 서비스 계정과 연결한 다음, 이 서비스 계정을 **TaskRun** 또는 **PipelineRun** 리소스와 연결합니다.

프로세스

1. 다음 명령을 입력하여 인증 정보가 포함된 기존 **config.json** 파일에서 컨테이너 레지스트리 인증 보안을 생성합니다.

```
$ oc create secret generic my-registry-credentials \ 1
  --from-file=config.json=/home/user/credentials/config.json 2
```

1 시크릿 이름(이 예에서는 **my-registry-credentials**)

2 **config.json** 파일의 경로 이름(이 예에서는 **/home/user/credentials/config.json**)

2. **serviceaccount.yaml** 파일에서 서비스 계정에 대한 YAML 매니페스트를 생성합니다. 이 매니페스트에서 보안을 서비스 계정과 연결합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: container-bot (1)
secrets:
  - name: my-registry-credentials (2)
```

(1) 서비스 계정의 이름입니다. 이 예에서는 **container-bot** 입니다.

(2) SSH 개인 키가 포함된 시크릿의 이름입니다. 예에서는 **my-registry-credentials**.

3. 작업 실행 또는 파이프라인 실행에 대한 YAML 매니페스트를 **run.yaml** 파일로 생성합니다. 이 파일에서 서비스 계정을 작업 실행 또는 파이프라인 실행과 연결합니다. 다음 예제 중 하나를 사용합니다.

   - 서비스 계정을 작업과 연결하려면 다음을 실행합니다.

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: build-container-task-run-2 (1)
spec:
  taskRunTemplate:
    serviceAccountName: container-bot (2)
  taskRef:
    name: build-container (3)
```

(1) 작업 실행의 이름입니다. 예에서는 **build-container-task-run-2** 입니다.

(2) 서비스 계정의 이름입니다. 이 예에서는 **container-bot** 입니다.

(3) 작업 이름입니다. 예에서는 **build-container** 입니다.

   - 서비스 계정을 파이프라인과 연결하려면 다음을 실행합니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: demo-pipeline (1)
  namespace: default
spec:
  taskRunTemplate:
    serviceAccountName: container-bot (2)
  pipelineRef:
    name: demo-pipeline (3)
```

(1) 파이프라인 실행의 이름입니다. 예에서는 **demo-pipeline** 입니다.

(2) 서비스 계정의 이름입니다. 이 예에서는 **container-bot** 입니다.

(3) 파이프라인의 이름입니다. 예에서는 **demo-pipeline** 입니다.

4. 다음 명령을 입력하여 변경 사항을 적용합니다.

```
$ oc apply --filename serviceaccount.yaml,run.yaml
```

5.2.5. 서비스 계정을 사용한 인증에 대한 추가 고려 사항

경우에 따라 서비스 계정을 사용하여 제공하는 인증 보안을 사용하려면 추가 단계를 완료해야 합니다.

5.2.5.1. 작업의 SSH Git 인증

작업 단계에서 Git 명령을 직접 호출하고 SSH 인증을 사용할 수 있지만 추가 단계를 완료해야 합니다.

OpenShift Pipelines는 **/tekton/home/.ssh** 디렉터리에 SSH 파일을 제공하고 **$HOME** 변수를 **/tekton/home** 으로 설정합니다. 그러나 Git SSH 인증은 **$HOME** 변수를 무시하고 사용자의 **/etc/passwd** 파일에 지정된 홈 디렉터리를 사용합니다. 따라서 Git 명령을 사용하는 단계에서는 **/tekton/home/.ssh** 디렉터리를 연결된 사용자의 홈 디렉터리에 심볼릭 링크해야 합니다.

예를 들어 작업이 **root** 사용자로 실행되는 경우 Git 명령 앞에 다음 명령을 포함해야 합니다.

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: example-git-task
spec:
  steps:
    - name: example-git-step
#     ...
      script:
        ln -s $HOME/.ssh /root/.ssh
#     ...
```

그러나 **git** 유형 또는 Tekton 카탈로그에서 사용할 수 있는 **git-clone** 작업의 파이프라인 리소스를 사용할 때는 명시적 심볼릭 링크가 필요하지 않습니다.

git 유형 작업에서 SSH 인증을 사용하는 예로, authenticating-git-commands.yaml 을 참조하십시오.

5.2.5.2. 루트가 아닌 사용자로 시크릿 사용

다음과 같은 특정 시나리오에서 루트가 아닌 사용자로 시크릿을 사용해야 할 수 있습니다.

- 컨테이너가 실행을 실행하는 데 사용하는 사용자 및 그룹은 플랫폼에 의해 무작위화됩니다.
- 작업의 단계에서는 루트가 아닌 보안 컨텍스트를 정의합니다.
- 작업은 작업의 모든 단계에 적용되는 글로벌 루트가 아닌 보안 컨텍스트를 지정합니다.

이러한 시나리오에서는 작업 실행 및 파이프라인 실행을 루트가 아닌 사용자로 실행할 때 다음 측면을 고려하십시오.

- Git에 대한 SSH 인증을 사용하려면 사용자에게 **/etc/passwd** 디렉터리에 유효한 홈 디렉터리가 구성되어 있어야 합니다. 유효한 홈 디렉터리가 없는 UID를 지정하면 인증에 실패합니다.
- SSH 인증은 **$HOME** 환경 변수를 무시합니다. 따라서 OpenShift Pipelines(**/tekton/home**)에서 루트가 아닌 사용자의 유효한 홈 디렉터리에 정의된 **$HOME** 디렉터리의 적절한 시크릿 파일을 심볼릭해야 합니다.

또한 루트가 아닌 보안 컨텍스트에서 SSH 인증을 구성하려면 예제의 **git-clone-and-check** 단계를 참조하십시오.

5.3. 작업 영역을 사용하여 보안 제공

작업 공간을 사용하여 Git 리포지토리 및 컨테이너 리포지토리에 대한 인증을 위한 시크릿을 제공할 수 있습니다.

작업에서 이름이 지정된 작업 영역을 구성하여 작업 영역을 마운트된 경로를 지정할 수 있습니다. 작업을 실행할 때 이 이름이 있는 작업 공간으로 시크릿을 제공합니다. OpenShift Pipelines가 작업을 실행하면 시크릿의 정보를 작업에 사용할 수 있습니다.

작업 영역을 사용하여 인증 보안을 제공하는 경우 시크릿에 대한 주석이 필요하지 않습니다.

5.3.1. 작업 영역을 사용하여 Git에 대한 SSH 인증 구성

SSH 키를 사용하여 구성된 리포지토리에서 리소스를 검색하려면 파이프라인에 대한 SSH 기반 인증을 구성해야 합니다.

파이프라인에 대한 SSH 기반 인증을 구성하려면 SSH 개인 키를 사용하여 인증 시크릿을 생성하고, 이 시크릿에 대해 이름이 지정된 작업 영역을 구성하고, 작업을 실행할 때 시크릿을 지정합니다.

프로세스

1. 다음 명령을 입력하여 기존 **.ssh** 디렉터리의 파일에서 Git SSH 인증 보안을 생성합니다.

```
$ oc create secret generic my-github-ssh-credentials \ 1
  --from-file=id_ed25519=/home/user/.ssh/id_ed25519 \ 2
  --from-file=known_hosts=/home/user/.ssh/known_hosts 3
```

1. 시크릿 이름(이 예에서 **my-github-ssh-credentials**)
2. 개인 키 파일의 이름 및 전체 경로 이름(이 예에서는 **/home/user/.ssh/id_ed25519**)
3. 알려진 호스트 파일의 이름 및 전체 경로 이름(이 예제에서는 **/home/user/.ssh/known_hosts**)

2. 작업 정의에서 Git 인증을 위해 이름이 지정된 작업 공간을 구성합니다(예: **ssh-directory**):

   작업 공간 정의 예

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: git-clone
spec:
  workspaces:
    - name: ssh-directory
      description: |
        A .ssh directory with private key, known_hosts, config, etc.
```

3. 작업 단계에서 **$(workspaces.<workspace_name>.path)** 환경 변수의 경로를 사용하여 디렉터리에 액세스합니다(예: **$(workspaces.ssh-directory.path)**).

4. 작업을 실행할 때 **tkn task start** 명령에 **--workspace** 인수를 포함하여 이름이 지정된 작업 공간에 대한 보안을 지정합니다.

```
$ tkn task start <task_name>
    --workspace name=<workspace_name>,secret=<secret_name> (1)
    # ...
```

(1) & **lt;workspace_name** >을 구성한 작업 공간 이름으로 바꾸고 < **secret_name** >을 생성한 시크릿 이름으로 바꿉니다.

인증에 **SSH** 키를 사용하여 **Git** 리포지토리 복제 작업 예

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: git-clone
spec:
  workspaces:
    - name: output
      description: The git repo will be cloned onto the volume backing this Workspace.
    - name: ssh-directory
      description: |
        A .ssh directory with private key, known_hosts, config, etc. Copied to
        the user's home before git commands are executed. Used to authenticate
        with the git remote when performing the clone. Binding a Secret to this
        Workspace is strongly recommended over other volume types
  params:
    - name: url
      description: Repository URL to clone from.
      type: string
    - name: revision
      description: Revision to checkout. (branch, tag, sha, ref, etc...)
      type: string
      default: ""
    - name: gitInitImage
      description: The image providing the git-init binary that this Task runs.
      type: string
      default: "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init:v0.37.0"
  results:
    - name: commit
      description: The precise commit SHA that was fetched by this Task.
    - name: url
      description: The precise URL that was fetched by this Task.
  steps:
    - name: clone
      image: "$(params.gitInitImage)"
      script: |
        #!/usr/bin/env sh
        set -eu
        # This is necessary for recent version of git
        git config --global --add safe.directory '*'
        cp -R "$(workspaces.ssh-directory.path)" "${HOME}"/.ssh (1)
        chmod 700 "${HOME}"/.ssh
        chmod -R 400 "${HOME}"/.ssh/*
```

```
    CHECKOUT_DIR="$(workspaces.output.path)/"
    /ko-app/git-init \
      -url="$(params.url)" \
      -revision="$(params.revision)" \
      -path="${CHECKOUT_DIR}"
    cd "${CHECKOUT_DIR}"
    RESULT_SHA="$(git rev-parse HEAD)"
    EXIT_CODE="$?"
    if [ "${EXIT_CODE}" != 0 ] ; then
      exit "${EXIT_CODE}"
    fi
    printf "%s" "${RESULT_SHA}" > "$(results.commit.path)"
    printf "%s" "$(params.url)" > "$(results.url.path)"
```

1 이 스크립트는 ssh가 자격 증명을 검색하는 표준 폴더인 **${HOME}/. ssh** 에 시크릿 콘텐츠를 복사합니다.

작업 실행을 위한 명령 예

```
$ tkn task start git-clone
    --param url=git@github.com:example-github-user/buildkit-tekton
    --workspace name=output,emptyDir=""
    --workspace name=ssh-directory,secret=my-github-ssh-credentials
    --use-param-defaults --showlog
```

5.3.2. 작업 영역을 사용하여 컨테이너 레지스트리 인증 구성

레지스트리에서 컨테이너 이미지를 검색하려면 파이프라인의 경우 해당 레지스트리에 대한 인증을 구성해야 합니다.

컨테이너 레지스트리에 대한 인증을 구성하려면 Docker 구성 파일을 사용하여 인증 보안을 생성하고, 작업에서 이 시크릿에 대해 이름이 지정된 작업 영역을 구성하고, 작업을 실행할 때 시크릿을 지정합니다.

프로세스

1. 다음 명령을 입력하여 인증 정보가 포함된 기존 **config.json** 파일에서 컨테이너 레지스트리 인증 보안을 생성합니다.

```
$ oc create secret generic my-registry-credentials \ 1
  --from-file=config.json=/home/user/credentials/config.json 2
```

1 시크릿 이름(이 예에서는 **my-registry-credentials**)

2 **config.json** 파일의 경로 이름(이 예에서는 **/home/user/credentials/config.json**)

2. 작업 정의에서 Git 인증을 위해 이름이 지정된 작업 공간을 구성합니다(예: **ssh-directory**):

작업 공간 정의 예

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
```

```
  name: skopeo-copy
spec:
  workspaces:
    - name: dockerconfig
      description: Includes a docker `config.json`
# ...
```

3. 작업 단계에서 **$(workspaces.<workspace_name>.path)** 환경 변수의 경로를 사용하여 디렉터리에 액세스합니다(예: **$(workspaces.dockerconfig.path)**.

4. 작업을 실행하려면 **tkn task start** 명령에 **--workspace** 인수를 포함하여 이름이 지정된 작업 공간에 대한 보안을 지정합니다.

```
$ tkn task start <task_name>
    --workspace name=<workspace_name>,secret=<secret_name> (1)
    # ...
```

(1) & **lt;workspace_name** >을 구성한 작업 공간 이름으로 바꾸고 < **secret_name** >을 생성한 시크릿 이름으로 바꿉니다.

Skopeo를 사용하여 컨테이너 리포지토리에서 이미지를 복사하는 작업의 예

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: skopeo-copy
spec:
  workspaces:
    - name: dockerconfig (1)
      description: Includes a docker `config.json`
  steps:
    - name: clone
      image: quay.io/skopeo/stable:v1.8.0
      env:
      - name: DOCKER_CONFIG
        value: $(workspaces.dockerconfig.path) (2)
      script: |
        #!/usr/bin/env sh
        set -eu
        skopeo copy docker://docker.io/library/ubuntu:latest
docker://quay.io/example_repository/ubuntu-copy:latest
```

(1) **config.json** 파일이 포함된 작업 공간의 이름입니다.

(2) **DOCKER_CONFIG** 환경 변수는 **dockerconfig** 작업 공간에 있는 **config.json** 파일의 위치를 가리킵니다. Skopeo는 이 환경 변수를 사용하여 인증 정보를 가져옵니다.

작업 실행을 위한 명령 예

```
$ tkn task start skopeo-copy
    --workspace name=dockerconfig,secret=my-registry-credentials
    --use-param-defaults --showlog
```

5.3.3. 작업 영역을 사용하여 특정 단계로 시크릿 제한

작업 영역을 사용하여 인증 보안을 제공하고 작업에서 작업 영역을 정의하는 경우 기본적으로 작업의 모든 단계에서 작업 영역을 사용할 수 있습니다.

보안을 특정 단계로 제한하려면 작업 사양과 단계 사양 모두에서 작업 공간을 정의합니다.

프로세스

- 다음 예제와 같이 작업 사양 및 단계 사양 모두에서 **workspaces:** 정의를 추가합니다.

  하나의 단계만 인증 정보 작업 공간에 액세스할 수 있는 작업 정의의 예

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: git-clone-build
spec:
  workspaces: (1)
    - name: ssh-directory
      description: |
        A .ssh directory with private key, known_hosts, config, etc.
# ...
  steps:
    - name: clone
      workspaces: (2)
        - name: ssh-directory
# ...
    - name: build (3)
# ...
```

(1) 작업 사양의 **ssh-directory** 작업 공간에 대한 정의입니다.

(2) 단계 사양의 **ssh-directory** 작업 공간에 대한 정의입니다. 인증 정보는 이 단계에서 **$(workspaces.ssh-directory.path)** 디렉터리로 사용할 수 있습니다.

(3) 이 단계에는 **ssh-directory** 작업 공간에 대한 정의가 포함되어 있지 않으므로 이 단계에서 인증 정보를 사용할 수 없습니다.

6장. BUILDAH를 루트가 아닌 사용자로 사용하여 컨테이너 이미지 빌드

컨테이너에서 root 사용자로 OpenShift Pipelines를 실행하면 컨테이너 프로세스 및 호스트를 다른 잠재적으로 악의적인 리소스에 노출할 수 있습니다. 컨테이너에서 루트가 아닌 특정 사용자로 워크로드를 실행하여 이러한 유형의 노출을 줄일 수 있습니다.

대부분의 경우 이 작업에서 이미지를 빌드하고 사용자 네임스페이스를 구성하는 사용자 지정 작업을 생성하여 root 권한 없이 Buildah를 실행할 수 있습니다.

이 구성을 사용하여 이미지가 성공적으로 빌드되지 않는 경우 사용자 정의 서비스 계정(SA) 및 SCC(보안 컨텍스트 제약 조건) 정의를 사용할 수 있지만 이 옵션을 사용하는 경우 Buildah 단계를 활성화하여 권한을 부여해야 합니다(**allowPrivilegeEscalation: true**).

6.1. 사용자 네임스페이스를 구성하여 루트가 아닌 사용자로 BUILDAH 실행

사용자 네임스페이스를 구성하는 것은 루트가 아닌 사용자로 작업에서 Buildah를 실행하는 가장 간단한 방법입니다. 그러나 일부 이미지는 이 옵션을 사용하여 빌드하지 못할 수 있습니다.

사전 요구 사항

- **oc** 명령줄 유틸리티를 설치했습니다.

프로세스

1. **openshift-pipelines** 네임스페이스에 제공된 **buildah** 작업의 사본을 생성하고 복사 이름을 **buildah-as-user** 로 변경하려면 다음 명령을 입력합니다.

```
$ oc get task buildah -n openshift-pipelines -o yaml | yq '. |= (del .metadata |= with_entries(select(.key == "name" )))' | yq '.kind="Task"' | yq '.metadata.name="buildah-as-user"' | oc create -f -
```

2. 다음 명령을 입력하여 복사한 **buildah** 작업을 편집합니다.

```
$ oc edit task buildah-as-user
```

다음 예와 같이 새 작업에서 **주석** 및 **stepTemplate** 섹션을 생성합니다.

buildah-as-user 작업에 추가된 예

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  annotations:
    io.kubernetes.cri-o.userns-mode: 'auto:size=65536;map-to-root=true'
    io.openshift.builder: 'true'
  name: assemble-containerimage
  namespace: pipeline-namespace
spec:
  description: This cluster task builds an image.
# ...
  stepTemplate:
    env:
```

```
    - name: HOME
      value: /tekton/home
  image: $(params.builder-image)
  imagePullPolicy: IfNotPresent
  name: ''
  resources:
    limits:
      cpu: '1'
      memory: 4Gi
    requests:
      cpu: 100m
      memory: 2Gi
  securityContext:
    capabilities:
      add:
        - SETFCAP
    runAsNonRoot: true
    runAsUser: 1000 (1)
  workingDir: $(workspaces.working-directory.path)
# ...
```

1 **podTemplate** 이 사용되므로 **runAsUser:** 설정은 엄격하게 필요하지 않습니다.

3. 새 **buildah-as-user** 작업을 사용하여 파이프라인에 이미지를 빌드합니다.

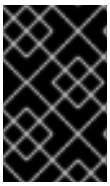
6.2. 사용자 지정 SA 및 SCC를 정의하여 ROOT가 아닌 사용자로 BUILDAH 실행

Buildah를 루트가 아닌 사용자로 사용하여 컨테이너 이미지의 빌드를 실행하려면 다음 단계를 수행할 수 있습니다.

- 사용자 정의 서비스 계정(SA) 및 SCC(보안 컨텍스트 제약 조건)를 정의합니다.
- ID **1000** 과 함께 **build** 사용자를 사용하도록 Buildah를 구성합니다.
- 사용자 정의 구성 맵으로 작업 실행을 시작하거나 파이프라인 실행과 통합합니다.

6.2.1. 사용자 정의 서비스 계정 및 보안 컨텍스트 제약 조건 구성

기본 **파이프라인** SA에서는 네임스페이스 범위 외부의 사용자 ID를 사용할 수 있습니다. 기본 SA에 대한 종속성을 줄이기 위해 사용자 ID **1000** 을 사용하여 **빌드** 사용자에 필요한 클러스터 역할 및 역할 바인딩을 사용하여 사용자 지정 SA 및 SCC를 정의할 수 있습니다.



중요

현재 컨테이너에서 Buildah를 성공적으로 실행하려면 **allowPrivilegeEscalation** 설정을 활성화해야 합니다. 이 설정을 통해 Buildah는 루트가 아닌 사용자로 실행할 때 **SETUID** 및 **SETGID** 기능을 활용할 수 있습니다.

프로세스

- 필요한 클러스터 역할 및 역할 바인딩을 사용하여 사용자 지정 SA 및 SCC를 만듭니다.

예: 사용된 ID 1000의 사용자 정의 SA 및 SCC

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pipelines-sa-userid-1000 (1)
---
kind: SecurityContextConstraints
metadata:
  annotations:
  name: pipelines-scc-userid-1000 (2)
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true (3)
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
- KILL
runAsUser: (4)
  type: MustRunAs
  uid: 1000
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-userid-1000-clusterrole (5)
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - pipelines-scc-userid-1000
```

```
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pipelines-scc-userid-1000-rolebinding 6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-userid-1000-clusterrole
subjects:
- kind: ServiceAccount
  name: pipelines-sa-userid-1000
```

1. 사용자 지정 SA를 정의합니다.
2. 수정된 **runAsUser** 필드를 사용하여 제한된 권한을 기반으로 생성된 사용자 정의 SCC를 정의합니다.
3. 현재 컨테이너에서 Buildah를 성공적으로 실행하려면 **allowPrivilegeEscalation** 설정을 활성화해야 합니다. 이 설정을 통해 Buildah는 루트가 아닌 사용자로 실행할 때 **SETUID** 및 **SETGID** 기능을 활용할 수 있습니다.
4. 사용자 ID **1000** 으로 실행되도록 사용자 지정 SCC와 함께 연결된 모든 Pod를 제한합니다.
5. 사용자 지정 SCC를 사용하는 클러스터 역할을 정의합니다.
6. 사용자 지정 SCC를 사용하는 클러스터 역할을 사용자 지정 SA에 바인딩합니다.

6.2.2. 빌드 사용자를 사용하도록 Buildah 구성

Buildah 작업을 정의하여 사용자 ID **1000** 과 함께 **빌드** 사용자를 사용할 수 있습니다.

프로세스

1. **openshift-pipelines** 네임스페이스에 제공되는 **buildah** 작업의 사본을 생성합니다. 복사 이름을 **buildah-as-user** 로 변경합니다.

```
$ oc get task buildah -n openshift-pipelines -o yaml | yq '. |= (del .metadata |=
with_entries(select(.key == "name" )))' | yq '.kind="Task"' | yq '.metadata.name="buildah-as-
user"' | oc create -f -
```

2. 복사한 **buildah** 작업을 편집합니다.

```
$ oc edit task buildah-as-user
```

예: **build** 사용자를 사용하여 **Buildah** 작업

```
apiVersion: tekton.dev/v1
kind: Task
```

```
metadata:
  name: buildah-as-user
spec:
  description: >-
    Buildah task builds source into a container image and
    then pushes it to a container registry.
    Buildah Task builds source into a container image using Project Atomic's
    Buildah build tool.It uses Buildah's support for building from Dockerfiles,
    using its buildah bud command.This command executes the directives in the
    Dockerfile to assemble a container image, then pushes that image to a
    container registry.
  params:
  - name: IMAGE
    description: Reference of the image buildah will produce.
  - name: BUILDER_IMAGE
    description: The location of the buildah builder image.
    default:
registry.redhat.io/rhel8/buildah@sha256:99cae35f40c7ec050fed3765b2b27e0b8bbea2aa2da7
c16408e2ca13c60ff8ee
  - name: STORAGE_DRIVER
    description: Set buildah storage driver
    default: vfs
  - name: DOCKERFILE
    description: Path to the Dockerfile to build.
    default: ./Dockerfile
  - name: CONTEXT
    description: Path to the directory to use as context.
    default: .
  - name: TLSVERIFY
    description: Verify the TLS on the registry endpoint (for push/pull to a non-TLS registry)
    default: "true"
  - name: FORMAT
    description: The format of the built container, oci or docker
    default: "oci"
  - name: BUILD_EXTRA_ARGS
    description: Extra parameters passed for the build command when building images.
    default: ""
  - description: Extra parameters passed for the push command when pushing images.
    name: PUSH_EXTRA_ARGS
    type: string
    default: ""
  - description: Skip pushing the built image
    name: SKIP_PUSH
    type: string
    default: "false"
  results:
  - description: Digest of the image just built.
    name: IMAGE_DIGEST
    type: string
  workspaces:
  - name: source
  steps:
  - name: build
    securityContext:
      runAsUser: 1000 (1)
    image: $(params.BUILDER_IMAGE)
```

```
    workingDir: $(workspaces.source.path)
    script: |
      echo "Running as USER ID `id`" 2
      buildah --storage-driver=$(params.STORAGE_DRIVER) bud \
        $(params.BUILD_EXTRA_ARGS) --format=$(params.FORMAT) \
        --tls-verify=$(params.TLSVERIFY) --no-cache \
        -f $(params.DOCKERFILE) -t $(params.IMAGE) $(params.CONTEXT)
      [[ "$(params.SKIP_PUSH)" == "true" ]] && echo "Push skipped" && exit 0
      buildah --storage-driver=$(params.STORAGE_DRIVER) push \
        $(params.PUSH_EXTRA_ARGS) --tls-verify=$(params.TLSVERIFY) \
        --digestfile $(workspaces.source.path)/image-digest $(params.IMAGE) \
        docker://$(params.IMAGE)
      cat $(workspaces.source.path)/image-digest | tee /tekton/results/IMAGE_DIGEST
    volumeMounts:
    - name: varlibcontainers
      mountPath: /home/build/.local/share/containers 3
  volumes:
  - name: varlibcontainers
    emptyDir: {}
```

1. Buildah 이미지의 **빌드** 사용자에게 해당하는 사용자 ID **1000** 으로 명시적으로 컨테이너를 실행합니다.
2. 사용자 ID를 표시하여 프로세스가 사용자 ID **1000** 으로 실행되고 있는지 확인합니다.
3. 필요에 따라 볼륨 마운트의 경로를 변경할 수 있습니다.

6.2.3. 사용자 정의 구성 맵 또는 파이프라인 실행을 사용하여 작업 실행 시작

사용자 정의 Buildah 클러스터 작업을 정의한 후 사용자 ID **1000** 을 사용하여 이미지를 빌드 사용자로 **빌드** 하는 **TaskRun** 오브젝트를 생성할 수 있습니다. 또한 **PipelineRun** 오브젝트의 일부로 **TaskRun** 오브젝트를 통합할 수 있습니다.

프로세스

1. 사용자 정의 **ConfigMap** 및 **Dockerfile** 오브젝트를 사용하여 **TaskRun** 오브젝트를 생성합니다.

   예: **Buildah**를 사용자 **ID 1000**으로 실행하는 작업 실행

```
apiVersion: v1
data:
  Dockerfile: |
    ARG BASE_IMG=registry.access.redhat.com/ubi9/ubi
    FROM $BASE_IMG AS buildah-runner
    RUN dnf -y update && \
        dnf -y install git && \
        dnf clean all
    CMD git
kind: ConfigMap
metadata:
  name: dockerfile 1
---
apiVersion: tekton.dev/v1
kind: TaskRun
```

```
metadata:
  name: buildah-as-user-1000
spec:
  taskRunTemplate:
    serviceAccountName: pipelines-sa-userid-1000 2
  params:
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  taskRef:
    kind: Task
    name: buildah-as-user
  workspaces:
  - configMap:
      name: dockerfile 3
    name: source
```

1. Dockerfile이 있는 일부 소스를 가져오는 이전 작업 없이 작업에 중점을 두고 있으므로 구성 맵을 사용합니다.
2. 생성한 서비스 계정의 이름입니다.
3. **buildah-as-user** 작업의 소스 작업 공간으로 구성 맵을 마운트합니다.

2. (선택 사항) 파이프라인 및 해당 파이프라인 실행을 생성합니다.

   예: 파이프라인 및 해당 파이프라인 실행

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-buildah-as-user-1000
spec:
  params:
  - name: IMAGE
  - name: URL
  workspaces:
  - name: shared-workspace
  - name: sslcertdir
    optional: true
  tasks:
  - name: fetch-repository 1
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
```

```
    - name: URL
      value: $(params.URL)
    - name: SUBDIRECTORY
      value: ""
    - name: DELETE_EXISTING
      value: "true"
  - name: buildah
    taskRef:
      name: buildah-as-user 2
    runAfter:
    - fetch-repository
    workspaces:
    - name: source
      workspace: shared-workspace
    - name: sslcertdir
      workspace: sslcertdir
    params:
    - name: IMAGE
      value: $(params.IMAGE)
---
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipelinerun-buildah-as-user-1000
spec:
  taskRunSpecs:
    - pipelineTaskName: buildah
      taskServiceAccountName: pipelines-sa-userid-1000 3
  params:
  - name: URL
    value: https://github.com/openshift/pipelines-vote-api
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  pipelineRef:
    name: pipeline-buildah-as-user-1000
  workspaces:
  - name: shared-workspace 4
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 100Mi
```

1. **git-clone** 클러스터 작업을 사용하여 Dockerfile이 포함된 소스를 가져와서 수정된 Buildah 작업을 사용하여 빌드합니다.
2. 수정된 Buildah 작업을 참조합니다.
3. Buildah 작업에 대해 생성한 서비스 계정을 사용합니다.
4. 컨트롤러에서 자동으로 생성한 PVC(영구 볼륨 클레임)를 사용하여 **git-clone** 작업과 수정된 Buildah 작업 간에 데이터를 공유합니다.

3. 작업 실행 또는 파이프라인 실행을 시작합니다.

6.3. 권한이 없는 빌드의 제한

권한이 없는 빌드의 프로세스는 대부분의 **Dockerfile** 오브젝트에서 작동합니다. 그러나 몇 가지 알려진 제한 사항으로 인해 빌드가 실패할 수 있습니다.

- 필요한 권한 문제 부족으로 인해 **--mount=type=cache** 옵션을 사용하면 실패할 수 있습니다. 자세한 내용은 이 문서를 참조하십시오.
- 마운트 리소스에 사용자 정의 SCC에서 제공하지 않는 추가 기능이 필요하므로 **--mount=type=secret** 옵션을 사용하면 실패합니다.

추가 리소스

- SCC(보안 컨텍스트 제약 조건) 관리