Red Hat OpenStack Platform 13

컨테이너화된 서비스로 전환

OpenStack Platform 컨테이너화된 서비스 작업을 위한 기본 가이드

Last Updated: 2023-05-25

Red Hat OpenStack Platform 13 컨테이너화된 서비스로 전환

OpenStack Platform 컨테이너화된 서비스 작업을 위한 기본 가이드

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

초록

이 가이드에서는 사용자가 컨테이너에서 실행되는 OpenStack Platform 서비스를 익숙한 작업을 수행할 수 있도록 지원하는 몇 가지 기본 정보를 제공합니다.

차례

1장. 소개

이전 버전의 Red Hat OpenStack Platform은 Systemd로 관리되는 서비스를 사용했습니다. 그러나 최신 버전의 OpenStack Platform에서는 이제 컨테이너를 사용하여 서비스를 실행합니다. 일부 관리자는 컨테이너화된 OpenStack Platform 서비스가 작동하는 방식을 잘 이해하지 못하기 때문에 이 안내서에서는 OpenStack Platform 컨테이너 이미지 및 컨테이너화된 서비스를 이해하는 데 도움이 됩니다. 여기에는 다음이 포함됩니다.

- 컨테이너 이미지를 가져오고 수정하는 방법
- 오버클라우드에서 컨테이너화된 서비스를 관리하는 방법
- 컨테이너가 Systemd 서비스와 어떻게 다른지 이해

주요 목표는 컨테이너화된 OpenStack Platform 서비스에 대한 충분한 지식을 습득하여 Systemd 기반 환경에서 컨테이너 기반 환경으로 전환하는 것입니다.

1.1. 컨테이너화된 서비스 및 KOLLA

각 주요 RHOSP(Red Hat OpenStack Platform) 서비스는 컨테이너에서 실행됩니다. 이를 통해 각 서비스를 호스트와 분리된 자체 네임스페이스 내에서 분리한 상태로 유지할 수 있습니다. 다음과 같은 효과가 있습니다.

- 배포 중에 RHOSP는 Red Hat 고객 포털에서 컨테이너 이미지를 가져와서 실행합니다.
- **podman** 명령은 서비스 시작 및 중지와 같은 관리 기능을 수행합니다.
- 컨테이너를 업그레이드하려면 새 컨테이너 이미지를 가져와 기존 컨테이너를 최신 버전으로 교체해야 합니다.

Red Hat OpenStack Platform은 **Kolla** 툴셋으로 빌드 및 관리되는 컨테이너 세트를 사용합니다.

2장. 컨테이너 이미지 가져오기 및 수정

컨테이너화된 오버클라우드는 필요한 컨테이너 이미지가 있는 레지스트리에 액세스해야 합니다. 이 장에서는 Red Hat OpenStack Platform에 컨테이너 이미지를 사용하도록 레지스트리 및 오버클라우드 구성을 준비하는 방법에 대해 설명합니다.

이 가이드에서는 레지스트리를 사용하도록 오버클라우드를 구성하는 몇 가지 사용 사례를 제공합니다. 이러한 사용 사례 중 하나를 시도하기 전에 이미지 준비 명령을 사용하는 방법을 숙지하는 것이 좋습니다. 자세한 내용은 2.2절. "컨테이너 이미지 준비 명령 사용법"를 참조하십시오.

2.1. 레지스트리 방법

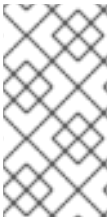
Red Hat OpenStack Platform은 다음 레지스트리 유형을 지원합니다.

원격 레지스트리

오버클라우드는 **registry.redhat.io** 에서 직접 컨테이너 이미지를 가져옵니다. 이 방법은 초기 구성을 생성하는 가장 쉬운 방법입니다. 그러나 각 오버클라우드 노드는 Red Hat Container Catalog에서 각 이미지를 직접 가져오므로 네트워크 혼잡 및 느린 배포가 발생할 수 있습니다. 또한 모든 오버클라우드 노드에는 Red Hat Container Catalog에 대한 인터넷 액세스가 필요합니다.

로컬 레지스트리

언더클라우드는 **docker-distribution** 서비스를 사용하여 레지스트리 역할을 합니다. 이를 통해 director는 **registry.redhat.io** 의 이미지를 동기화하고 **docker-distribution** 레지스트리로 내보낼 수 있습니다. 오버클라우드를 생성할 때 오버클라우드는 언더클라우드의 **docker-distribution** 레지스트리에서 컨테이너 이미지를 가져옵니다. 이 방법을 사용하면 내부 레지스트리를 저장할 수 있으므로 배포를 가속화하고 네트워크 혼잡을 줄일 수 있습니다. 그러나 언더클라우드는 기본 레지스트리 역할을 하며 컨테이너 이미지에 대해 제한된 라이프사이클 관리 기능을 제공합니다.



참고

docker -distribution 서비스는 docker와 별도로 작동합니다. **Docker** 는 이미지를 **docker-distribution** 레지스트리로 가져와 푸시하는 데 사용되며 오버클라우드에 이미지를 제공하지 않습니다. 오버클라우드는 **docker-distribution** 레지스트리에서 이미지를 가져옵니다.

Satellite Server

컨테이너 이미지의 전체 애플리케이션 라이프사이클을 관리하고 Red Hat Satellite 6 서버를 통해 게시합니다. 오버클라우드는 Satellite 서버에서 이미지를 가져옵니다. 이 방법을 사용하면 Red Hat OpenStack Platform 컨테이너를 저장, 관리 및 배포할 수 있는 엔터프라이즈급 솔루션을 제공합니다.

목록에서 방법을 선택하고 레지스트리 세부 정보를 계속 구성합니다.

참고

다중 아키텍처 클라우드를 빌드하는 경우 로컬 레지스트리 옵션은 지원되지 않습니다.

2.2. 컨테이너 이미지 준비 명령 사용법

이 섹션에서는 명령의 다양한 옵션에 대한 개념 정보를 포함하여 **openstack overcloud container image prepare** 명령을 사용하는 방법에 대한 개요를 제공합니다.

오버클라우드의 컨테이너 이미지 환경 파일 생성

openstack overcloud container image prepare 명령의 주요 사용 중 하나는 오버클라우드에서 사용하

는 이미지 목록이 포함된 환경 파일을 생성하는 것입니다. **openstack overcloud deploy** 와 같은 오버클라우드 배포 명령을 사용하여 이 파일을 추가합니다. **openstack overcloud container image prepare** 명령에서는 이 함수에 다음 옵션을 사용합니다.

--output-env-file

결과 환경 파일 이름을 정의합니다.

다음 스니펫은 이 파일의 내용입니다.

```
parameter_defaults:
  DockerAodhApiImage: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
  DockerAodhConfigImage: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
...
```

환경 파일에는 언더클라우드 레지스트리의 IP 주소 및 포트로 설정된 **DockerInsecureRegistryAddress** 매개변수도 포함되어 있습니다. 이 매개변수는 SSL/TLS 인증서가 없는 언더클라우드 레지스트리에서 이미지에 액세스하도록 오버클라우드 노드를 구성합니다.

가져오기 방법용 컨테이너 이미지 목록 생성

OpenStack Platform 컨테이너 이미지를 다른 레지스트리 소스로 가져오려는 경우 이미지 목록을 생성할 수 있습니다. list 구문은 주로 컨테이너 이미지를 언더클라우드의 컨테이너 레지스트리로 가져오는 데 사용되지만, Red Hat Satellite 6과 같은 다른 가져오기 방법에 맞게 이 목록의 형식을 수정할 수 있습니다.

openstack overcloud container image prepare 명령에서는 이 함수에 다음 옵션을 사용합니다.

--output-images-file

가져오기 목록에 대한 결과 파일 이름을 정의합니다.

다음은 이 파일의 예입니다.

```
container_images:
- imagename: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
- imagename: registry.redhat.io/rhosp13/openstack-aodh-evaluator:13.0-34
...
```

컨테이너 이미지의 네임스페이스 설정

--output-env-file 및 **--output-images-file** 옵션 모두 결과 이미지 위치를 생성하기 위해 네임스페이스가 필요합니다. **openstack overcloud container image prepare** 명령에서는 다음 옵션을 사용하여 가져올 컨테이너 이미지의 소스 위치를 설정합니다.

--namespace

컨테이너 이미지의 네임스페이스를 정의합니다. 일반적으로 디렉터리가 있는 호스트 이름 또는 IP 주소입니다.

--prefix

이미지 이름 앞에 추가할 접두사를 정의합니다.

결과적으로 director는 다음 형식을 사용하여 이미지 이름을 생성합니다.

- **[NAMESPACE]/[PREFIX][IMAGE NAME]**

컨테이너 이미지 태그 설정

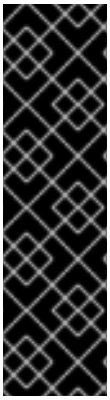
--tag 및 **--tag-from-label** 옵션을 함께 사용하여 각 컨테이너 이미지의 태그를 설정합니다.

--tag

소스의 모든 이미지에 대한 특정 태그를 설정합니다. 이 옵션만 사용하는 경우 director는 이 태그를 사용하여 모든 컨테이너 이미지를 가져옵니다. 그러나 이 옵션을 **--tag-from-label** 와 함께 사용하는 경우 director는 **--tag** 를 소스 이미지로 사용하여 라벨에 따라 특정 버전 태그를 식별합니다. **--tag** 옵션은 기본적으로 **latest** 로 설정됩니다.

--tag-from-label

지정된 컨테이너 이미지 레이블의 값을 사용하여 모든 이미지의 버전 지정된 태그를 검색하고 가져옵니다. director는 **--tag** 에 설정된 값으로 태그가 지정된 각 컨테이너 이미지를 검사한 다음 컨테이너 이미지 레이블을 사용하여 director가 레지스트리에서 가져오는 새 태그를 구성합니다. 예를 들어 **--tag-from-label {version}-{release}** 를 설정하면 director는 **version** 및 **release** 레이블을 사용하여 새 태그를 구성합니다. 하나의 컨테이너의 경우 **version** 을 **13.0** 으로 설정할 수 있으며 **릴리스** 를 **34** 로 설정할 수 있으며 이로 인해 태그가 **13.0-34** 가 됩니다.



중요

Red Hat Container Registry는 특정 버전 형식을 사용하여 모든 Red Hat OpenStack Platform 컨테이너 이미지에 태그를 지정합니다. 이 버전 형식은 **{version}-{release}** 이며, 각 컨테이너 이미지는 컨테이너 메타데이터의 라벨로 저장됩니다. 이 버전 형식은 하나의 **{release}** 에서 다음 버전으로의 업데이트를 용이하게 하는 데 도움이 됩니다. 따라서 **openstack overcloud container image prepare** 명령을 실행할 때 항상 **--tag-from-label {version}-{release}** 를 사용해야 합니다. 컨테이너 이미지를 가져오기 위해 자체적으로 **--tag** 를 사용하지 마십시오. 예를 들어 **--tag latest** 를 자체적으로 사용하면 director에서 컨테이너 이미지를 업데이트하기 위해 태그를 변경해야 하므로 업데이트를 수행할 때 문제가 발생합니다.

2.3. 추가 서비스를 위한 컨테이너 이미지

director는 핵심 OpenStack Platform 서비스에 대해서만 컨테이너 이미지를 준비합니다. 일부 추가 기능은 추가 컨테이너 이미지가 필요한 서비스를 사용합니다. 환경 파일을 사용하여 이러한 서비스를 활성화합니다. **openstack overcloud container image prepare** 명령에서는 다음 옵션을 사용하여 환경 파일과 해당 컨테이너 이미지를 포함합니다.

-e

추가 컨테이너 이미지를 활성화하는 환경 파일을 포함합니다.

다음 표에서는 컨테이너 이미지와 해당 환경 파일 위치를 **/usr/share/openstack-tripleo-heat-templates** 디렉터리 내의 해당 환경 파일 위치를 사용하는 추가 서비스 샘플 목록을 제공합니다.

서비스	환경 파일
Ceph Storage	**environments/ceph-ansible/ceph-ansible.yaml**
collectd	**environments/services-docker/collectd.yaml**
의회	**environments/services-docker/congress.yaml**
fluentd	**environments/services-docker/fluentd.yaml**
OpenStack Bare Metal(ironic)	**environments/services-docker/ironic.yaml**

서비스	환경 파일
OpenStack Data Processing(sahara)	**environments/services-docker/sahara.yaml**
OpenStack EC2-API	**environments/services-docker/ec2-api.yaml**
OpenStack Key Manager(barbican)	**environments/services-docker/barbican.yaml**
OpenStack Load Balancing-as-a-Service(octavia)	**environments/services-docker/octavia.yaml**
OpenStack Shared File System Storage(manila)	**environments/manila-{backend-name}-config.yaml** 참고: 자세한 내용은 OpenStack Shared File System(manila)을 참조하십시오.
OVN(Open Virtual Network)	**environments/services-docker/neutron-ovn-dvr-ha.yaml**
Sensu	**environments/services-docker/sensu-client.yaml**

다음 몇 섹션에서는 추가 서비스를 포함하는 예제를 제공합니다.

Ceph Storage

오버클라우드와 함께 Red Hat Ceph Storage 클러스터를 배포하는 경우 **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** 환경 파일을 포함해야 합니다. 이 파일을 사용하면 오버클라우드에서 구성 가능 컨테이너화된 서비스를 활성화할 수 있으며, 이미지를 준비하기 위해 director에서 이러한 서비스가 활성화되었는지 확인해야 합니다.

이 환경 파일 외에도 OpenStack Platform 서비스와 다른 Ceph Storage 컨테이너 위치도 정의해야 합니다. **--set** 옵션을 사용하여 Ceph Storage와 관련된 다음 매개변수를 설정합니다.

--set ceph_namespace

Ceph Storage 컨테이너 이미지의 네임스페이스를 정의합니다. 이 기능은 **--namespace** 옵션과 유사합니다.

--set ceph_image

Ceph Storage 컨테이너 이미지의 이름을 정의합니다. 일반적으로 **rhceph-3-rhel7** 입니다.

--set ceph_tag

Ceph Storage 컨테이너 이미지에 사용할 태그를 정의합니다. 이 기능은 **--tag** 옵션과 유사합니다. **--tag-from-label** 이 지정되면 이 태그부터 versioned 태그가 검색됩니다.

다음 스니펫은 컨테이너 이미지 파일에 Ceph Storage를 포함하는 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml \
  --set ceph_namespace=registry.redhat.io/rhceph \
```

```
  --set ceph_image=rhceph-3-rhel7 \
  --tag-from-label {version}-{release} \
  ...
```

OpenStack Bare Metal(ironic)

오버클라우드에 OpenStack Bare Metal(ironic)을 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml** 환경 파일을 포함해야 합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml \
  ...
```

OpenStack Data Processing(sahara)

오버클라우드에 OpenStack Data Processing(sahara)을 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/sahara.yaml** 환경 파일을 포함해야 합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/sahara.yaml \
  ...
```

OpenStack Neutron SR-IOV

오버클라우드에 OpenStack Neutron SR-IOV를 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-sriov.yaml** 환경 파일을 포함합니다. 기본 컨트롤러 및 컴퓨팅 역할은 SR-IOV 서비스를 지원하지 않으므로 **-r** 옵션을 사용하여 SR-IOV 서비스가 포함된 사용자 지정 역할 파일을 포함해야 합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -r ~/custom_roles_data.yaml
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-sriov.yaml \
  ...
```

OpenStack Load Balancing-as-a-Service(octavia)

오버클라우드에 OpenStack Load Balancing-as-a-Service를 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml** 환경 파일을 포함합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml
\
  ...
```

OpenStack Shared File System(manila)

manila-{backend-name}-config.yaml 형식을 사용하여 지원되는 백엔드를 선택하여 해당 백엔드와 함께 공유 파일 시스템을 배포할 수 있습니다. 공유 파일 시스템 서비스 컨테이너는 다음 환경 파일을 포함하여 준비할 수 있습니다.

```
environments/manila-isilon-config.yaml
environments/manila-netapp-config.yaml
environments/manila-vmax-config.yaml
environments/manila-cephfsnative-config.yaml
environments/manila-cephfsganesha-config.yaml
environments/manila-unity-config.yaml
environments/manila-vnx-config.yaml
```

환경 파일 사용자 정의 및 배포에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- *공유 파일 시스템 서비스 의 NFS 백엔드 가이드를 통해 CephFS 에 업데이트된 환경 배포*
- 공유 파일 시스템 서비스의 NetApp 백엔드 *가이드에 있는 NetApp 백엔드와 공유 파일 시스템 서비스*배포
- 공유 파일 시스템 서비스의 CephFS 백엔드 *가이드를 사용하여 공유 파일 시스템 서비스*배포

2.4. RED HAT 레지스트리를 원격 레지스트리 소스로 사용

Red Hat은 **registry.redhat.io** 에서 오버클라우드 컨테이너 이미지를 호스팅합니다. 원격 레지스트리에서 이미지를 가져오는 것은 레지스트리가 이미 구성되어 있고 필요한 모든 항목은 가져올 이미지의 URL 및 네임스페이스이기 때문에 가장 간단한 방법입니다. 하지만 오버클라우드를 생성하는 동안 오버클라우드 노드는 모두 원격 리포지토리에서 이미지를 가져오므로 외부 연결을 수행할 수 있습니다. 따라서 이 방법은 프로덕션 환경에 권장되지 않습니다. 프로덕션 환경의 경우 다음 방법 중 하나를 사용합니다.

- 로컬 레지스트리 설정
- Red Hat Satellite 6에서 이미지 호스트

절차

1. 오버클라우드 배포의 **registry.redhat.io** 에서 직접 이미지를 가져오려면 이미지 매개변수를 지정해야 합니다. 다음 명령을 실행하여 컨테이너 이미지 환경 파일을 생성합니다.

```
(undercloud) $ sudo openstack overcloud container image prepare \
  --namespace=registry.redhat.io/rhosp13 \
  --prefix=openstack- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

   - **-e** 옵션을 사용하여 선택적 서비스에 대한 환경 파일을 포함합니다.
   - 사용자 지정 역할 파일을 포함하려면 **-r** 옵션을 사용합니다.
   - Ceph Storage를 사용하는 경우 Ceph Storage 컨테이너 이미지 위치 **--set ceph_namespace**,**--set ceph_image**,**--set ceph_tag** 를 정의하는 추가 매개변수를 포함합니다.

2. **overcloud_images.yaml** 파일을 수정하고 배포 중에 **registry.redhat.io** 로 인증하려면 다음 매개변수를 포함합니다.

```
ContainerImageRegistryLogin: true
ContainerImageRegistryCredentials:
  registry.redhat.io:
    <USERNAME>: <PASSWORD>
```

- &lt **;USERNAME** > 및 < **PASSWORD** >를 **registry.redhat.io** 의 인증 정보로 바꿉니다. **overcloud_images.yaml** 파일에는 언더클라우드의 이미지 위치가 포함되어 있습니다. 배포에 이 파일을 포함합니다.

참고

openstack overcloud deploy 명령을 실행하기 전에 원격 레지스트리에 로그인해야 합니다.

```
(undercloud) $ sudo docker login registry.redhat.io
```

레지스트리 구성이 준비되었습니다.

2.5. 언더클라우드를 로컬 레지스트리로 사용

오버클라우드 컨테이너 이미지를 저장하도록 언더클라우드에 로컬 레지스트리를 설정할 수 있습니다.

director를 사용하여 **registry.redhat.io** 에서 각 이미지를 가져오고 각 이미지를 언더클라우드에서 실행되는 **docker-distribution** 레지스트리로 내보낼 수 있습니다. director를 사용하여 오버클라우드 생성 프로세스 중에 오버클라우드를 생성하면 노드가 언더클라우드 **docker-distribution** 레지스트리에서 관련 이미지를 가져옵니다.

이렇게 하면 외부 네트워크 연결을 설정하지 않고 배포 프로세스를 가속화할 수 있는 내부 네트워크 내에 컨테이너 이미지의 네트워크 트래픽을 유지합니다.

절차

1. 로컬 언더클라우드 레지스트리의 주소를 찾습니다. 이 주소는 다음 패턴을 사용합니다.

```
<REGISTRY_IP_ADDRESS>:8787
```

   이전에 **undercloud.conf** 파일의 **local_ip** 매개변수로 설정한 언더클라우드의 IP 주소를 사용합니다. 아래 명령의 경우 주소는 **192.168.24.1:8787** 이라고 가정합니다.

2. **registry.redhat.io** 에 로그인합니다.

```
(undercloud) $ docker login registry.redhat.io --username $RH_USER --password
$RH_PASSWD
```

3. 이미지를 로컬 레지스트리에 업로드하는 템플릿을 생성하고 해당 이미지를 참조하는 환경 파일을 생성합니다.

```
(undercloud) $ openstack overcloud container image prepare \
  --namespace=registry.redhat.io/rhosp13 \
  --push-destination=192.168.24.1:8787 \
  --prefix=openstack- \
```

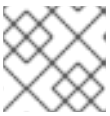
```
--tag-from-label {version}-{release} \
--output-env-file=/home/stack/templates/overcloud_images.yaml \
--output-images-file /home/stack/local_registry_images.yaml
```

- **-e** 옵션을 사용하여 선택적 서비스에 대한 환경 파일을 포함합니다.
- 사용자 지정 역할 파일을 포함하려면 **-r** 옵션을 사용합니다.
- Ceph Storage를 사용하는 경우 Ceph Storage 컨테이너 이미지 위치 **--set ceph_namespace**,**--set ceph_image**,**--set ceph_tag** 를 정의하는 추가 매개변수를 포함합니다.

4. 다음 두 파일이 생성되었는지 확인합니다.
   - 원격 소스의 컨테이너 이미지 정보가 포함된 **local_registry_images.yaml** 이 파일을 사용하여 Red Hat Container Registry(**registry.redhat.io**)에서 언더클라우드로 이미지를 가져옵니다.
   - **overcloud_images.yaml** 은 언더클라우드의 최종 이미지 위치가 포함된 overcloud_images.yaml입니다. 배포에 이 파일을 포함합니다.
5. 원격 레지스트리에서 컨테이너 이미지를 가져와서 언더클라우드 레지스트리로 푸시합니다.

```
(undercloud) $ openstack overcloud container image upload \
 --config-file  /home/stack/local_registry_images.yaml \
 --verbose
```

필요한 이미지를 가져오는 데는 네트워크와 언더클라우드 디스크의 속도에 따라 시간이 걸릴 수 있습니다.



참고

컨테이너 이미지는 약 10GB의 디스크 공간을 사용합니다.

6. 이제 이미지가 언더클라우드의 **docker-distribution** 레지스트리에 저장됩니다. 언더클라우드의 **docker-distribution** 레지스트리의 이미지 목록을 보려면 다음 명령을 실행합니다.

```
(undercloud) $  curl http://192.168.24.1:8787/v2/_catalog | jq .repositories[]
```

참고

_catalog 리소스는 100개의 이미지만 표시합니다. 더 많은 이미지를 표시하려면 **_catalog** 리소스가 포함된 **?n=<interger** > 쿼리 문자열을 사용하여 더 많은 수의 이미지를 표시합니다.

```
(undercloud) $  curl http://192.168.24.1:8787/v2/_catalog?n=150 | jq
.repositories[]
```

특정 이미지의 태그 목록을 보려면 **skopeo** 명령을 사용합니다.

```
(undercloud) $ curl -s http://192.168.24.1:8787/v2/rhosp13/openstack-keystone/tags/list | jq
.tags
```

태그가 지정된 이미지를 확인하려면 **skopeo** 명령을 사용합니다.

```
(undercloud) $ skopeo inspect --tls-verify=false
docker://192.168.24.1:8787/rhosp13/openstack-keystone:13.0-44
```

레지스트리 구성이 준비되었습니다.

2.6. SATELLITE 서버를 레지스트리로 사용

Red Hat Satellite 6는 레지스트리 동기화 기능을 제공합니다. 이를 통해 여러 이미지를 Satellite 서버로 가져와 애플리케이션 라이프사이클의 일부로 관리할 수 있습니다. Satellite는 다른 컨테이너 활성화 시스템이 사용할 레지스트리 역할도 합니다. 컨테이너 이미지 관리 방법에 대한 자세한 내용은 *Red Hat Satellite 6 Content Management Guide*의 "Managing Container Images"를 참조하십시오.

다음 절차의 예제에서는 Red Hat Satellite 6용 **hammer** 명령행 툴과 **ACME**라는 조직을 사용합니다. 이 조직을 실제로 사용하는 Satellite 6 조직으로 대체하십시오.

절차

1. 이미지를 로컬 레지스트리로 가져올 템플릿을 생성합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud container image prepare \
  --namespace=rhosp13 \
  --prefix=openstack- \
  --output-images-file /home/stack/satellite_images
```

   - **-e** 옵션을 사용하여 선택적 서비스에 대한 환경 파일을 포함합니다.
   - 사용자 지정 역할 파일을 포함하려면 **-r** 옵션을 사용합니다.
   - Ceph Storage를 사용하는 경우 Ceph Storage 컨테이너 이미지 위치 **--set ceph_namespace**,**--set ceph_image**,**--set ceph_tag** 를 정의하는 추가 매개변수를 포함합니다.

참고

이 버전의 **openstack overcloud container image prepare** 명령은 이미지 목록을 생성하기 위해 **registry.redhat.io** 의 레지스트리를 대상으로 합니다. 이후 단계에서 사용된 **openstack overcloud container image prepare** 명령과 다른 값을 사용합니다.

2. 이렇게 하면 컨테이너 이미지 정보가 있는 **satellite_images** 파일이 생성됩니다. 이 파일을 사용하여 컨테이너 이미지를 Satellite 6 서버와 동기화합니다.

3. **satellite_images** 파일에서 YAML 관련 정보를 제거하고 이미지 목록만 포함하는 플랫 파일로 변환합니다. 다음 **sed** 명령은 이 작업을 수행합니다.

```
(undercloud) $ awk -F ':' '{if (NR!=1) {gsub("[[:space:]]", ""); print $2}}' ~/satellite_images >
~/satellite_images_names
```

   이를 통해 Satellite 서버로 가져온 이미지 목록이 제공됩니다.

4. **satellite_images_names** 파일을 Satellite 6 **hammer** 툴이 포함된 시스템으로 복사합니다. 또는 *Hammer CLI 가이드*의 지침을 사용하여 **hammer** 툴을 언더클라우드에 설치합니다.

5. 다음 **hammer** 명령을 실행하여 Satellite 조직에 새 제품(**OSP13 Containers**)을 생성합니다.

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP13 Containers"
```

   이 사용자 지정 제품에 이미지를 저장합니다.

6. 제품에 기본 컨테이너 이미지를 추가합니다.

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP13 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhosp13/openstack-base \
  --name base
```

7. **satellite_images** 파일에서 오버클라우드 컨테이너 이미지를 추가합니다.

```
$ while read IMAGE; do \
  IMAGENAME=$(echo $IMAGE | cut -d"/" -f2 | sed "s/openstack-//g" | sed "s/:.*//g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP13 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE \
  --name $IMAGENAME ; done < satellite_images_names
```

8. 컨테이너 이미지를 동기화합니다.

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP13 Containers"
```

   Satellite 서버가 동기화를 완료할 때까지 기다립니다.

참고

설정에 따라 **hammer**에서 Satellite 서버 사용자 이름과 암호가 필요할 수 있습니다. **hammer**를 구성한 후 구성 파일을 사용하여 자동으로 로그인할 수 있습니다. *Hammer CLI 가이드의 "Authentication" 섹션을 참조하십시오*.

9. Satellite 6 서버에서 콘텐츠 뷰를 사용하는 경우 새 콘텐츠 뷰 버전을 생성하여 이미지를 통합합니다.

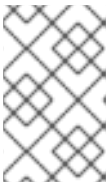
10. **기본** 이미지에 사용 가능한 태그를 확인합니다.

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --product "OSP13 Containers"
```

그러면 OpenStack Platform 컨테이너 이미지의 태그가 표시됩니다.

11. 언더클라우드로 돌아가서 Satellite 서버에서 이미지에 대한 환경 파일을 생성합니다. 다음은 환경 파일을 생성하는 예제 명령입니다.

```
(undercloud) $ openstack overcloud container image prepare \
  --namespace=satellite6.example.com:5000 \
  --prefix=acme-osp13_containers- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```



참고

이 버전의 **openstack overcloud container image prepare** 명령에서는 Satellite 서버를 대상으로 합니다. 이전 단계에서 사용된 **openstack overcloud 컨테이너 이미지 prepare** 명령과 다른 값을 사용합니다.

이 명령을 실행할 때 다음 데이터를 포함합니다.

- **--namespace** - Satellite 서버의 레지스트리 URL 및 포트입니다. Red Hat Satellite의 레지스트리 포트는 5000입니다. 예를 들어 **--namespace=satellite6.example.com:5000** 입니다.

참고

Red Hat Satellite 버전 6.10을 사용하는 경우 포트를 지정할 필요가 없습니다. 기본 포트 **443** 이 사용됩니다. 자세한 내용은 " RHOSP13 배포를 Red Hat Satellite 6.10에 어떻게 적용할 수 있습니까?"에서 참조하십시오.

- **--prefix=** - 접두사는 소문자 문자를 사용하고 밑줄로 대체되는 라벨의 Satellite 6 규칙을 기반으로 합니다. 접두사는 콘텐츠 뷰 사용 여부에 따라 달라집니다.
  - 콘텐츠 뷰를 사용하는 경우 구조는 **[org]-[environment]-[content view]-[product]-**입니다. 예: **acme-production-myosp13-osp13_containers-**.
  - 콘텐츠 뷰를 사용하지 않는 경우 구조는 **[org]-[product]-**입니다. 예: **acme-osp13_containers-**.
- **--tag-from-label {version}-{release}** - 각 이미지의 최신 태그를 식별합니다.
- **-e** - 선택적 서비스에 대한 환경 파일을 포함합니다.
- **-r** - 사용자 지정 역할 파일을 포함합니다.
- **--set ceph_namespace**,**--set ceph_image**,**--set ceph_tag** - Ceph Storage를 사용하는 경우 Ceph Storage 컨테이너 이미지 위치를 정의하는 추가 매개변수를 포함합니다. 이제 **ceph_image**에는 Satellite별 접두사가 포함됩니다. 이 접두사는 **--prefix** 옵션과 동일한 값입니다. 예를 들면 다음과 같습니다.

```
--set ceph_image=acme-osp13_containers-rhceph-3-rhel7
```

이렇게 하면 오버클라우드에서 Satellite 이름 지정 규칙을 사용하여 Ceph 컨테이너 이미지를 사용할 수 있습니다.

12. **overcloud_images.yaml** 파일에는 Satellite 서버의 이미지 위치가 포함되어 있습니다. 배포에 이 파일을 포함합니다.

레지스트리 구성이 준비되었습니다.

2.7. 컨테이너 이미지 수정

Red Hat은 Red Hat Container Catalog(**registry.redhat.io**)를 통해 사전 빌드된 컨테이너 이미지 세트를 제공합니다. 이러한 이미지를 수정하고 추가 레이어를 추가할 수 있습니다. 이는 인증된 타사 드라이버에 RPM을 컨테이너에 추가하는 데 유용합니다.

참고

수정된 OpenStack Platform 컨테이너 이미지를 지속적으로 지원하려면 결과 이미지가 "Red Hat Container Support Policy" 를 준수하도록 합니다.

이 예에서는 최신 **openstack-keystone** 이미지를 사용자 지정하는 방법을 보여줍니다. 그러나 이러한 지침은 다른 이미지에도 적용할 수 있습니다.

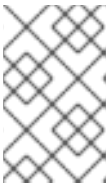
절차

1. 수정하려는 이미지를 가져옵니다. 예를 들어 **openstack-keystone** 이미지의 경우 다음을 실행합니다.

```
$ sudo docker pull registry.redhat.io/rhosp13/openstack-keystone:latest
```

2. 원본 이미지에서 기본 사용자를 확인합니다. 예를 들어 **openstack-keystone** 이미지의 경우 다음을 실행합니다.

```
$ sudo docker run -it registry.redhat.io/rhosp13/openstack-keystone:latest whoami
root
```



참고

openstack-keystone 이미지는 **root** 를 기본 사용자로 사용합니다. 다른 이미지는 특정 사용자를 사용합니다. 예를 들어 **openstack-glance-api** 는 기본 사용자에 **glance** 를 사용합니다.

3. **Dockerfile** 을 생성하여 기존 컨테이너 이미지에서 추가 계층을 빌드합니다. 다음은 Container Catalog에서 최신 OpenStack Identity(keystone) 이미지를 가져와서 이미지에 사용자 지정 RPM 파일을 설치하는 예입니다.

```
FROM registry.redhat.io/rhosp13/openstack-keystone
MAINTAINER Acme
LABEL name="rhosp13/openstack-keystone-acme" vendor="Acme" version="2.1"
release="1"

# switch to root and install a custom RPM, etc.
USER root
COPY custom.rpm /tmp
```

```
RUN rpm -ivh /tmp/custom.rpm

# switch the container back to the default user
USER root
```

4. 새 이미지를 빌드하고 태그를 지정합니다. 예를 들어 **/home/stack/keystone** 디렉터리에 저장된 로컬 **Dockerfile** 로 빌드하고 언더클라우드의 로컬 레지스트리에 태그를 지정하려면 다음을 수행합니다.

```
$ docker build /home/stack/keystone -t "192.168.24.1:8787/rhosp13/openstack-keystone-acme:rev1"
```

5. 결과 이미지를 언더클라우드의 로컬 레지스트리로 푸시합니다.

```
$ docker push 192.168.24.1:8787/rhosp13/openstack-keystone-acme:rev1
```

6. 오버클라우드 컨테이너 이미지 환경 파일(일반적으로 **overcloud_images.yaml**)을 편집하고 사용자 정의 컨테이너 이미지를 사용하도록 적절한 매개변수를 변경합니다.

주의

Container Catalog는 완전한 소프트웨어 스택을 사용하여 컨테이너 이미지를 게시합니다. Container Catalog에서 업데이트 및 보안 수정으로 컨테이너 이미지를 릴리스하면 기존 사용자 정의 컨테이너에 이러한 업데이트가 **포함되지 않으므로** 카탈로그의 새 이미지 버전을 사용하여 다시 빌드해야 합니다.

3장. 컨테이너로 오버클라우드 배포 및 업데이트

이 장에서는 컨테이너 기반 오버클라우드를 생성하고 계속 업데이트하는 방법에 대한 정보를 제공합니다.

3.1. 오버클라우드 배포

다음 절차에서는 최소 구성으로 오버클라우드를 배포하는 방법을 설명합니다. 결과적으로 기본 2-노드 오버클라우드(1 컨트롤러 노드, 컴퓨팅 노드 1개)가 됩니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. **배포** 명령을 실행하고 오버클라우드 이미지 위치(일반적으로 **overcloud_images.yaml**)가 포함된 파일을 포함합니다.

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/overcloud_images.yaml \
  --ntp-server pool.ntp.org
```

3. 오버클라우드가 배포를 완료할 때까지 기다립니다.

3.2. 오버클라우드 업데이트

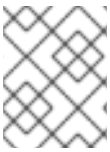
컨테이너화된 오버클라우드 업데이트에 대한 자세한 내용은 *Red Hat OpenStack Platform Updated 가이드*를 참조하십시오.

4장. 컨테이너화된 서비스 작업

이 장에서는 컨테이너를 관리하는 명령과 OpenStack Platform 컨테이너 문제를 해결하는 방법에 대한 몇 가지 예를 제공합니다.

4.1. 컨테이너화된 서비스 관리

오버클라우드는 컨테이너에서 대부분의 OpenStack Platform 서비스를 실행합니다. 호스트의 개별 서비스를 제어해야 하는 경우도 있습니다. 이 섹션에서는 오버클라우드 노드에서 실행하여 컨테이너화된 서비스를 관리할 수 있는 일반적인 **docker** 명령을 제공합니다. Docker를 사용하여 컨테이너를 **관리하는** 방법에 대한 자세한 내용은 *컨테이너 시작하기 가이드의 Docker 형식 컨테이너 작업 을* 참조하십시오.



참고

이러한 명령을 실행하기 전에 오버클라우드 노드에 로그인되어 있고 언더클라우드에서 이 명령을 실행하지 않는지 확인합니다.

컨테이너 및 이미지 목록 표시

실행 중인 컨테이너를 나열하려면 다음을 수행합니다.

```
$ sudo docker ps
```

중지된 컨테이너 또는 실패한 컨테이너도 나열하려면 **--all** 옵션을 추가합니다.

```
$ sudo docker ps --all
```

컨테이너 이미지를 나열하려면 다음을 수행합니다.

```
$ sudo docker images
```

컨테이너 속성 확인

컨테이너 또는 컨테이너 이미지의 속성을 보려면 **docker inspect** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 검사하려면 다음을 수행합니다.

```
$ sudo docker inspect keystone
```

기본 컨테이너 작업 관리

컨테이너화된 서비스를 다시 시작하려면 **docker restart** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 다시 시작하려면 다음을 수행합니다.

```
$ sudo docker restart keystone
```

컨테이너화된 서비스를 중지하려면 **docker stop** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 중지하려면 다음을 수행합니다.

```
$ sudo docker stop keystone
```

중지된 컨테이너화된 서비스를 시작하려면 **docker start** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 시작하려면 다음을 수행합니다.

-

```
$ sudo docker start keystone
```

참고

컨테이너를 다시 시작한 후에 컨테이너 내의 서비스 설정 파일에 대한 모든 변경 사항을 되돌립니다. 컨테이너가 **/var/lib/config-data/puppet-generated/** 에서 노드의 로컬 파일 시스템에 있는 파일을 기반으로 서비스 구성을 다시 생성하기 때문입니다. 예를 들어 **keystone** 컨테이너 내의 **/etc/keystone/keystone.conf**를 편집하고 컨테이너를 다시 시작하는 경우 컨테이너가 노드의 로컬 파일 시스템에서 **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf**를 사용하여 구성을 다시 생성합니다. 이는 다시 시작하기 전에 컨테이너 내에 만들어진 모든 변경 사항을 덮어씁니다.

컨테이너 모니터링

컨테이너화된 서비스의 로그를 확인하려면 **docker logs** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너의 로그를 보려면 다음을 수행합니다.

```
$ sudo docker logs keystone
```

컨테이너 액세스

컨테이너화된 서비스의 쉘에 들어가려면 **docker exec** 명령을 사용하여 **/bin/bash** 를 시작합니다. 예를 들어 **keystone** 컨테이너 쉘에 들어가려면 다음을 수행합니다.

```
$ sudo docker exec -it keystone /bin/bash
```

root 사용자로 **keystone** 컨테이너 쉘을 시작하려면 다음을 수행합니다.

```
$ sudo docker exec --user 0 -it <NAME OR ID> /bin/bash
```

컨테이너를 종료하려면 다음을 수행합니다.

```
# exit
```

4.2. 컨테이너화된 서비스 문제 해결

오버클라우드 배포 중 또는 이후에 컨테이너화된 서비스가 실패하는 경우 다음 권장 사항을 사용하여 실패의 근본 원인을 확인합니다.

참고

이러한 명령을 실행하기 전에 오버클라우드 노드에 로그인되어 있고 언더클라우드에서 이 명령을 실행하지 않는지 확인합니다.

컨테이너 로그 확인

각 컨테이너는 메인 프로세스의 표준 출력을 유지합니다. 이 출력은 컨테이너 실행 중에 실제로 수행되는 작업을 확인하는 데 도움이 되는 로그 역할을 합니다. 예를 들어 **keystone** 컨테이너의 로그를 보려면 다음 명령을 사용합니다.

```
$ sudo docker logs keystone
```

대부분의 경우 이 로그는 컨테이너 실패의 원인을 제공합니다.

컨테이너 검사

컨테이너 정보를 확인해야 하는 경우가 있습니다. 예를 들어 다음 명령을 사용하여 **keystone** 컨테이너 데이터를 확인합니다.

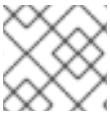
```
$ sudo docker inspect keystone
```

낮은 수준의 구성 데이터를 포함하는 JSON 오브젝트를 제공합니다. 출력을 **jq** 명령으로 전달하여 특정 데이터를 구문 분석할 수 있습니다. 예를 들어 **keystone** 컨테이너에 대한 컨테이너 마운트를 보려면 다음 명령을 실행합니다.

```
$ sudo docker inspect keystone | jq .[0].Mounts
```

또한 **--format** 옵션을 사용하여 단일 행에 대한 데이터를 구문 분석할 수 있으며 이는 컨테이너 데이터 세트에 대해 명령을 실행할 때 유용합니다. 예를 들어 **keystone** 컨테이너 실행에 사용된 옵션을 재생성하려면 **--format** 옵션과 함께 다음 **inspect** 명령을 사용합니다.

```
$ sudo docker inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}' keystone
```



참고

--format 옵션은 쿼리를 생성할 때 Go 구문을 사용합니다.

docker run 명령과 함께 이러한 옵션을 사용하여 문제 해결을 위해 컨테이너를 재생성합니다.

```
$ OPTIONS=$( sudo docker inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo docker run --rm $OPTIONS /bin/bash
```

컨테이너에서 명령 실행

특정 Bash 명령을 통해 컨테이너 내부 정보를 가져와야 하는 경우가 있습니다. 이 경우 다음 **docker** 명령을 사용하여 실행 중인 컨테이너 내에서 명령을 실행합니다. 예를 들어 **keystone** 컨테이너에서 명령을 실행하려면 다음을 실행합니다.

```
$ sudo docker exec -ti keystone <COMMAND>
```

참고

-ti 옵션은 대화식 가상 터미널(pseudoterminal)에서 명령을 실행합니다.

&lt **;COMMAND&gt**;를 원하는 명령으로 교체합니다. 예를 들어 각 컨테이너에는 서비스 연결을 확인하기 위한 상태 점검 스크립트가 있습니다. 다음 명령을 사용하여 **keystone**에 대해 상태 점검 스크립트를 실행할 수 있습니다.

```
$ sudo docker exec -ti keystone /openstack/healthcheck
```

컨테이너 쉘에 액세스하려면 명령으로 **/bin/bash** 를 사용하여 **docker exec** 를 실행합니다.

```
$ sudo docker exec -ti keystone /bin/bash
```

컨테이너 내보내기

컨테이너가 실패하면 파일의 전체 콘텐츠를 확인해야 합니다. 이 경우 컨테이너의 전체 파일 시스템을 **tar** 아카이브로 내보낼 수 있습니다. 예를 들어 **keystone** 컨테이너 파일 시스템을 내보내려면 다음 명령을 실행합니다.

```
$ sudo docker export keystone -o keystone.tar
```

이 명령은 **keystone.tar** 아카이브를 생성하여 추출 및 확인할 수 있습니다.

5장. SYSTEMD 서비스와 컨테이너화된 서비스 비교

이 장에서는 컨테이너화된 서비스가 Systemd 서비스와 어떻게 다른지를 보여주는 몇 가지 참조 자료를 제공합니다.

5.1. SYSTEMD 서비스 명령 VS 컨테이너화된 서비스 명령

다음 표는 Systemd 기반 명령과 Docker 동등한 간의 몇 가지 유사성을 보여줍니다. 이는 수행하려는 서비스 작업의 유형을 식별하는 데 도움이 됩니다.

함수	systemd 기반	Docker 기반
모든 서비스 나열	**systemctl list-units -t service**	**docker ps --all**
활성 서비스 나열	**systemctl list-units -t service --state active**	**Docker ps**
서비스 상태 확인	**systemctl status openstack-nova-api**	**docker ps --filter "name=nova_api$" --all**
서비스 중지	**systemctl stop openstack-nova-api**	**Docker stop nova_api**
서비스 시작	**systemctl start openstack-nova-api**	**Docker start nova_api**
서비스 재시작	**systemctl restart openstack-nova-api**	**Docker 재시작 nova_api**
서비스 구성 표시	**systemctl show openstack-nova-api** **systemctl cat openstack-nova-api**	**Docker inspect nova_api**
서비스 로그 표시	**journalctl -u openstack-nova-api**	**Docker 로그 nova_api**

5.2. SYSTEMD 서비스와 컨테이너화된 서비스 비교

다음 표는 Systemd 기반 OpenStack 서비스 및 컨테이너 기반 동등한 것을 보여줍니다.

OpenStack 서비스	systemd 서비스	Docker 컨테이너

OpenStack 서비스	systemd 서비스	Docker 컨테이너
aodh	**openstack-aodh-evaluator** **openstack-aodh-listener** **openstack-aodh-notifier** **httpd(openstack-aodh-api)**	**aodh_listener** **aodh_api** **aodh_notifier** **aodh_evaluator**
Ceilometer	**openstack-ceilometer-central** **openstack-ceilometer-collector** **openstack-ceilometer-notification** **httpd(openstack-ceilometer-api)**	**ceilometer_agent_notification** **ceilometer_agent_central**
cinder	**openstack-cinder-api** **openstack-cinder-scheduler** **openstack-cinder-volume**	**cinder_scheduler** **cinder_api** **openstack-cinder-volume-docker-0**
glance	**openstack-glance-api** **openstack-glance-registry**	**glance_api**
gnocchi	**openstack-gnocchi-metricd** **openstack-gnocchi-statsd** **httpd(openstack-gnocchi-api)**	**gnocchi_statsd** **gnocchi_api** **gnocchi_metricd**
heat	**openstack-heat-api-cfn** **openstack-heat-api-cloudwatch** **openstack-heat-api** **openstack-heat-engine**	**heat_api_cfn** **heat_engine** **heat_api**
Horizon	**httpd(openstack-dashboard)**	**Horizon**
keystone	**httpd(openstack-keystone)**	**keystone**

OpenStack 서비스	systemd 서비스	Docker 컨테이너
Neutron	**neutron-dhcp-agent** **neutron-l3-agent** **neutron-metadata-agent** **neutron-openvswitch-agent** **neutron-server**	**neutron_ovs_agent** **neutron_l3_agent** **neutron_metadata_agent** **neutron_dhcp** **neutron_api**
nova	**openstack-nova-api** **openstack-nova-conductor** **openstack-nova-consoleauth** **openstack-nova-novncproxy** **openstack-nova-scheduler**	**nova_metadata** **nova_api** **nova_conductor** **nova_vnc_proxy** **nova_consoleauth** **nova_api_cron** **nova_scheduler** **nova_placement**
panko		**panko_api**

OpenStack 서비스	systemd 서비스	Docker 컨테이너
swift	**openstack-swift-account-auditor** **openstack-swift-account-reaper** **openstack-swift-account-replicator** **openstack-swift-account** **openstack-swift-container-auditor** **openstack-swift-container-replicator** **openstack-swift-container-updater** **openstack-swift-container** **openstack-swift-object-auditor** **openstack-swift-object-expirer** **openstack-swift-object-replicator** **openstack-swift-object-updater** **openstack-swift-object** **openstack-swift-proxy**	**swift_proxy** **swift_account_server** **swift_container_auditor** **swift_object_expirer** **swift_object_updater** **swift_container_replicator** **swift_account_auditor** **swift_object_replicator** **swift_container_server** **swift_rsync** **swift_account_reaper** **swift_account_replicator** **swift_object_auditor** **swift_object_server** **swift_container_update**

5.3. SYSTEMD 로그 위치 및 컨테이너화된 로그 위치

다음 표는 Systemd 기반 OpenStack 로그와 컨테이너의 해당 로그를 보여줍니다. 모든 컨테이너 기반 로그 위치는 물리적 호스트에서 사용할 수 있으며 컨테이너에 마운트됩니다.

OpenStack 서비스	systemd 서비스 로그	Docker 컨테이너 로그
aodh	**/var/log/aodh/**	**/var/log/containers/aodh/** **/var/log/containers/httpd/aodh-api/**
Ceilometer	**/var/log/ceilometer/**	**/var/log/containers/ceilometer/**

OpenStack 서비스	systemd 서비스 로그	Docker 컨테이너 로그
cinder	**/var/log/cinder/**	**/var/log/containers/cinder/** **/var/log/containers/httpd/cinder-api/**
glance	**/var/log/glance/**	**/var/log/containers/glance/**
gnocchi	**/var/log/gnocchi/**	**/var/log/containers/gnocchi/** **/var/log/containers/httpd/gnocchi-api/**
heat	**/var/log/heat/**	**/var/log/containers/heat/** **/var/log/containers/httpd/heat-api/** **/var/log/containers/httpd/heat-api-cfn/**
Horizon	**/var/log/horizon/**	**/var/log/containers/horizon/** **/var/log/containers/httpd/horizon/**
keystone	**/var/log/keystone/**	**/var/log/containers/keystone** **/var/log/containers/httpd/keystone/**
데이터베이스	**/var/log/mariadb/** **/var/log/mongodb/** **/var/log/mysqld.log**	**/var/log/containers/mysql/**
Neutron	**/var/log/neutron/**	**/var/log/containers/neutron/** **/var/log/containers/httpd/neutron-api/**
nova	**/var/log/nova/**	**/var/log/containers/nova/** **/var/log/containers/httpd/nova-api/** **/var/log/containers/httpd/nova-placement/**

OpenStack 서비스	systemd 서비스 로그	Docker 컨테이너 로그
panko		**/var/log/containers/panko/** **/var/log/containers/httpd/panko-api/**
rabbitmq	**/var/log/rabbitmq/**	**/var/log/containers/rabbitmq/**
redis	**/var/log/redis/**	**/var/log/containers/redis/**
swift	**/var/log/swift/**	**/var/log/containers/swift/**

5.4. SYSTEMD 구성 VS 컨테이너화된 구성

다음 표는 Systemd 기반 OpenStack 구성 및 컨테이너의 해당 구성을 보여줍니다. 모든 컨테이너 기반 구성 위치는 물리적 호스트에서 컨테이너에 마운트되며 각 컨테이너 내의 구성으로 병합(**kolla**)됩니다.

OpenStack 서비스	systemd 서비스 구성	Docker 컨테이너 구성
aodh	**/etc/aodh/**	**/var/lib/config-data/puppet-generated/aodh/**
Ceilometer	**/etc/ceilometer/**	**/var/lib/config-data/puppet-generated/ceilometer/etc/ceilometer/**
cinder	**/etc/cinder/**	**/var/lib/config-data/puppet-generated/cinder/etc/cinder/**
glance	**/etc/glance/**	**/var/lib/config-data/puppet-generated/glance_api/etc/glance/**
gnocchi	**/etc/gnocchi/**	**/var/lib/config-data/puppet-generated/gnocchi/etc/gnocchi/**
haproxy	**/etc/haproxy/**	**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/**

OpenStack 서비스	systemd 서비스 구성	Docker 컨테이너 구성
heat	**/etc/heat/**	**/var/lib/config-data/puppet-generated/heat/etc/heat/** **/var/lib/config-data/puppet-generated/heat_api/etc/heat/** **/var/lib/config-data/puppet-generated/heat_api_cfn/etc/heat/**
Horizon	**/etc/openstack-dashboard/**	**/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/**
keystone	**/etc/keystone/**	**/var/lib/config-data/puppet-generated/keystone/etc/keystone/**
데이터베이스	**/etc/my.cnf.d/** **/etc/my.cnf**	**/var/lib/config-data/puppet-generated/mysql/etc/my.cnf.d/**
Neutron	**/etc/neutron/**	**/var/lib/config-data/puppet-generated/neutron/etc/neutron/**
nova	**/etc/nova/**	**/var/lib/config-data/puppet-generated/nova/etc/nova/** **/var/lib/config-data/puppet-generated/nova_placement/etc/nova/**
panko		**/var/lib/config-data/puppet-generated/panko/etc/panko**
rabbitmq	**/etc/rabbitmq/**	**/var/lib/config-data/puppet-generated/rabbitmq/etc/rabbitmq/**
redis	**/etc/redis/** **/etc/redis.conf**	**/var/lib/config-data/puppet-generated/redis/etc/redis/** **/var/lib/config-data/puppet-generated/redis/etc/redis.conf**

OpenStack 서비스	systemd 서비스 구성	Docker 컨테이너 구성
swift	**/etc/swift/**	**/var/lib/config-data/puppet-generated/swift/etc/swift/** **/var/lib/config-data/puppet-generated/swift_ringbuilder/etc/swift/**