2022년 전기 졸업과제 중간보고서

오픈소스 클라우드 플랫폼을 활용한 사용자 맞춤형 가상머신 관리 시스템

팀명: 뜬구름

201624540 이봉훈

201624406 고준성

201624444 김영후

지도교수: 염근혁(인)

목차

1. 과제의 목표

1.1. 과제 배경

최근 언택트 시대가 도래하면서 물리적 인프라를 직접 구축하지 않고도 컴퓨팅 자원에 접근할 수 있게 하는 클라우드 컴퓨팅이 각광받고 있다. 이러한 시대에 발맞춰 많은 클라우드 벤더들은 사용자에게 여러 유형의 클라우드 기반 서비스를 제공하고 있고, 클라우드 사용자들은 각자 다양한 목적과 용도로 이러한 서비스를 사용하고 있다. 그 중에서도 클라우드를 통해 사용자 각자의 요구에 알맞은 IT 인프라 자원을 이용하려는 사용자가 늘고 있다.

하지만 사용자가 클라우드 컴퓨팅을 활용하기 위해서는 관련 용어 및 가상머신에 대한 전문적인 지식이 요구되므로 클라우드 관련 전문 지식이 부족한 사용자는 가상머신에 대한 설정을 잘 다루지 못하는 문제가 생긴다. 또한 사용자의 요구에 부합하는 가상머신을 생성하기 위해서 하드웨어, 소프트웨어 등에 대한 요구사항을 받아들이고 처리할 수 있는 구조도 지원되어야 한다.

이와 더불어 클라우드 인프라 구축 후에 발생하는 관리적 측면의 어려움이 있다. 사용자의 클라우드 자원 활용에 따른 변경이 있을 경우 클라우드 자원의 활용도가 달라지게 되므로, 이를 처리하는 방안과 구조가 필요하다. 또한 갑작스러운 정전이나 재해 등과 같은 외부 요인 혹은 시스템 자체 결함 등의 내부 요인으로 클라우드 시스템에 장애가 생기는 경우, 클라우드 사용자가 장애발생에 유연하게 대처하는 것도 어려운 점 중 하나이다.

따라서 본 과제는 사용자의 요구사항을 입력 받아 클라우드 가상머신 생성 시 필요한 설정, 요구사항의 변경이 있을 시 가상머신의 업데이트, 재해발생 시 가상머신의 재해 복구를 지원하는 통합적인 가상 머신 관리 시스템 구현을 목표로 한다.

1.2. 과제 세부 목표

① 사용자 하드웨어, 소프트웨어 요구사항을 반영한 가상 머신 생성
② 가상머신의 하드웨어, 소프트웨어 요구사항 변경에 따른 맞춤형 가상 머신 관리
③ 재해복구(Disaster Recovery)기능을 갖춘 동기화된 멀티 클라우드 환경 구성

2. 요구조건 및 제약 사항 분석에 대한 수정사항

2.1. 요구조건 수정사항

아래 표 1은 착수보고서의 기능적 요구사항을 중간보고서의 유스케이스로 추출한 것을 나타낸 것이다.

표 1. 기능적 요구사항 - 유스케이스

기능적 요구사항	유스케이스 명
시스템 접근 제어	로그인
	로그아웃
	회원가입
	회원탈퇴
	토큰발급
사용자 요구사항 입력	템플릿 생성
맞춤형 Orchestration Template 생성	
사용자 가상환경 정보 출력	가상머신 관리
Template 기반 가상머신 생성 또는 업데이트	
재해 알림	- 백업용 가상머신 생성 - 가상환경 재해 및 복구 관리
알림 발생시 가상머신 전환	

2.2. 유스케이스 목록 및 명세

표 2 는 도출된 요구사항을 유스케이스로 구분한 것이다

표 2. 유스케이스 목록

유스케이스 ID	유스케이스 명
UC-001	로그인
UC-002	로그아웃
UC-003	회원가입
UC-004	회원탈퇴
UC-005	토큰 발급
UC-006	템플릿 생성
UC-007	가상머신 관리
UC-008	백업용 가상머신 생성
UC-009	가상환경 재해 및 복구 관리

표 3에서 표 11까지는 유스케이스에 대한 명세를 나타낸 것이다.

표 3. 유스케이스 명세 - 로그인

요구사항 ID	UC-001
유스케이스명	로그인
관련 액터	사용자, DB
개요	사용자가 가상머신 관리 시스템을 사용하기 위해 로그인한다.
선행조건	회원가입
이벤트 흐름	기본 흐름 1. 사용자가 아이디와 비밀번호를 입력한다. 2. 사용자가 로그인 버튼을 클릭한다. 3. 시스템은 사용자가 입력한 로그인 정보가 DB에 존재하는지 확인한다. 4. 로그인 정보가 DB에 존재하는 경우 클라우드 플랫폼에 사용자의 가상머신 정보를 요청한다. 5. 사용자 페이지로 이동 및 사용자의 가상머신 정보를 출력해준다. 대안 흐름 3A. 로그인 정보가 DB에 존재하지 않을 경우 정보 없음을 출력하고 로그인 창으로 이동한다.
후행조건	

표 4. 유스케이스 명세 - 로그아웃

요구사항 ID	UC-002
유스케이스명	로그아웃
관련 액터	사용자
개요	사용자가 가상머신 관리 시스템을 종료하기 위해 로그아웃한다.
선행조건	로그인
이벤트 흐름	기본 흐름 1. 사용자가 로그아웃 버튼을 클릭한다. 2. 시스템은 로그아웃 완료 화면을 띄우고 다시 로그인 화면으로 이동한다.
후행조건	

표 5. 유스케이스 명세 - 회원가입

요구사항 ID	UC-003
유스케이스명	회원가입
관련 액터	사용자, DB
개요	사용자가 시스템을 사용하기 위해 회원가입을 한다.
선행조건	
이벤트 흐름	기본 흐름 1. 사용자가 회원가입 버튼을 클릭한다. 2. 회원가입 화면으로 이동하여 사용자 정보(아이디, 비밀번호)를 입력한다. 3. 회원가입 완료 버튼을 클릭한다. 4. 시스템은 회원정보를 바탕으로 회원생성 후 회원가입 완료 메세지를 출력한다. 5. 시스템은 회원 정보를 확인하고 성공하면 사용자 페이지로 이동한다. 대안 흐름 4A. 이미 아이디가 존재하는 경우 회원가입 화면으로 돌아간다.
후행조건	

표 6. 유스케이스 명세 - 회원탈퇴

요구사항 ID	UC-004
유스케이스명	회원탈퇴
관련 액터	사용자, DB
개요	사용자가 시스템에서 회원탈퇴를 한다.
선행조건	로그인
이벤트 흐름	기본 흐름 1. 사용자가 탈퇴 버튼을 클릭한다. 2. 회원탈퇴 화면으로 이동하여 탈퇴 확인을 위해 비밀번호를 입력한다. 3. 회원탈퇴 확인 버튼을 클릭한다. 4. 시스템은 DB에서 해당 회원의 정보를 삭제한다. 5. 사용자에게 회원탈퇴 메세지를 출력해준다.
후행조건	

표 7. 유스케이스 명세 - 토큰 발급

요구사항 ID	UC-005
유스케이스명	토큰 발급
관련 액터	사용자, 클라우드 플랫폼
개요	시스템은 사용자의 인증을 위한 토큰을 발급한다.
선행조건	로그인
이벤트 흐름	기본흐름 1. 사용자가 시스템에 로그인한다. 2. 시스템은 클라우드 플랫폼에 토큰 발급을 요청한다. 3. 클라우드 플랫폼은 토큰을 생성한 후 시스템에 반환한다.
후행조건	

표 8. 유스케이스 명세 - 템플릿 생성

요구사항 ID	UC-006
유스케이스명	템플릿 생성
관련 액터	사용자
개요	시스템은 사용자가 입력한 요구사항을 기반으로 오케스트레이션 기능을 수행 할 수 있는 템플릿을 생성한다.
선행조건	로그인
이벤트 흐름	기본흐름 1. 사용자가 가상머신 생성 또는 업데이트 버튼을 누른다. 2. 가상머신 생성 또는 업데이트에 사용되는 템플릿 양식을 불러온다. 3. 템플릿의 각 항목에 사용자가 입력한 요구사항을 입력한다. 4. 모든 입력이 끝난 템플릿을 반환한다.
후행조건	

표 9. 유스케이스 명세 - 가상머신 관리

요구사항 ID	UC-007
유스케이스명	가상머신 관리
관련 액터	사용자, 클라우드 플랫폼
개요	시스템은 사용자의 가상머신 생성, 업데이트, 삭제 요청을 수행한다.
선행조건	로그인, 토큰 발급, 템플릿 생성
이벤트 흐름	기본흐름 1. 사용자가 가상머신 생성, 업데이트, 삭제 버튼을 누른다.
	가상머신 생성 또는 업데이트 흐름 2. 시스템은 사용자의 요구사항을 반영한 생성, 업데이트 템플릿을 생성한다. 3. 시스템은 요구사항 반영 템플릿을 바탕으로 각 클라우드 플랫폼에게 가상머신 생성 또는 업데이트를 요청한다. 4. 각 클라우드 플랫폼은 가상머신을 생성 또는 업데이트하고 결과를 반환한다. 5. 시스템은 생성 또는 업데이트 된 가상머신의 정보를 요청한다. 6. 각 클라우드 플랫폼은 가상머신의 정보를 반환한다.
	가상머신 삭제 흐름 2. 시스템은 각 클라우드 플랫폼에게 가상머신 삭제를 요청한다. 3. 각 클라우드 플랫폼은 가상머신을 삭제하고 결과를 반환한다. 4. 삭제 후 시스템은 완료 메세지를 출력한다.
후행조건	

표 10. 유스케이스 명세 - 백업용 가상머신 생성

요구사항 ID	UC-008
유스케이스명	백업용 가상머신 생성
관련액터	사용자, 클라우드 플랫폼
개요	시스템은 주기적으로 가상머신의 스냅샷을 생성한 후 이미지 형태로 베이스 클라우드 플랫폼 또는 백업용 클라우드 플랫폼에 저장한다.
선행조건	로그인, 토큰 발급, 가상머신 생성 또는 업데이트
이벤트 흐름	기본흐름 1. 사용자가 가상머신 생성 시 백업 주기를 입력한다. 2. 시스템은 주기적으로 베이스 클라우드 플랫폼에 가상머신의 백업 이미지, 가상머신 메타데이터 생성을 요청한다. 3. 베이스 클라우드 플랫폼은 가상머신의 백업 이미지, 가상머신 메타데이터를 생성하고 결과를 반환한다.
	베이스 클라우드 플랫폼의 백업 가상머신 생성 흐름 4.시스템은 베이스 클라우드 플랫폼에 백업 메타데이터를 기반으로 중지 상태의 가상머신 생성을 요청한다. 5. 베이스 클라우드 플랫폼에서 중지 상태의 가상머신을 생성하고 결과를 반환한다.
	백업용 클라우드 플랫폼의 백업 가상머신 생성 흐름 4. 생성된 백업 이미지를 시스템에 전송한다. 5. 시스템의 백업용 이미지 URL을 통해 백업용 클라우드 플랫폼에 중지상태의 가상머신을 생성한다. 6. 가상머신 생성 결과를 시스템에 반환한다.
후행조건	

표 11. 유스케이스 명세 - 가상환경 재해 및 복구 관리

요구사항 ID	UC-009
유스케이스명	가상환경 재해 및 복구 관리
관련액터	사용자, 클라우드 플랫폼
개요	시스템은 재해 사항, 재해 복구 상태를 모니터링하고 재해복구를 수행한다.
선행조건	로그인, 토큰 발급, 가상머신 생성 또는 업데이트
이벤트 흐름	기본흐름 1. 사용자가 가상머신을 생성 또는 업데이트한다. 2. 시스템은 주기적으로 가상머신 상태를 모니터링한다. 3. 가상머신 상태 이상이 감지되면 베이스 클라우드 플랫폼 상태를 조회한다. 대안흐름 2A. 가상머신 상태 이상이 감지되지 않으면 반복적으로 모니터링 모듈을 이용해 가상머신의 상태를 모니터링한다.
	베이스 클라우드 플랫폼 내 재해관리 흐름 4. 베이스 플랫폼 상태가 정상인 경우 베이스 클라우드 플랫폼 내 백업용 가상머신으로 환경 전환 이벤트를 발생시킨다. 5. 재해복구 관리 컴포넌트는 베이스 클라우드 플랫폼 내 백업 가상머신을 가동시킨다. 6. 베이스 클라우드 플랫폼의 백업 가상머신으로 사용환경을 전환한다.
	백업용 클라우드 플랫폼 내 재해관리 흐름 4. 베이스 클라우드 플랫폼 상태이상이 확인되면 백업용 클라우드 플랫폼으로 가상환경 전환 이벤트를 발생시킨다. 5. 재해복구 관리 컴포넌트는 이종 클라우드의 백업 가상머신을 가동시킨다. 6. 백업용 클라우드 플랫폼의 가상머신으로 사용환경을 전환한다. 7. 재해발생한 기존 클라우드 플랫폼의 상태를 주기적으로 모니터링 한다. 8. 상태 복구가 감지되면 가상머신 복구 이벤트를 발생시킨다. 9. 사용자가 백업용 클라우드 플랫폼에서 사용하던 가상머신의 이미지를 생성한다. 10. 생성된 이미지를 웹서버에 저장한다. 11. 웹서버에 저장된 이미지를 이용해 베이스 클라우드 플랫폼에서 가상머신을 생성한다. 12. 베이스 클라우드 플랫폼의 가상머신으로 사용환경을 전환한다.
후행조건	

3. 설계 상세화

3.1. Usecase Diagram

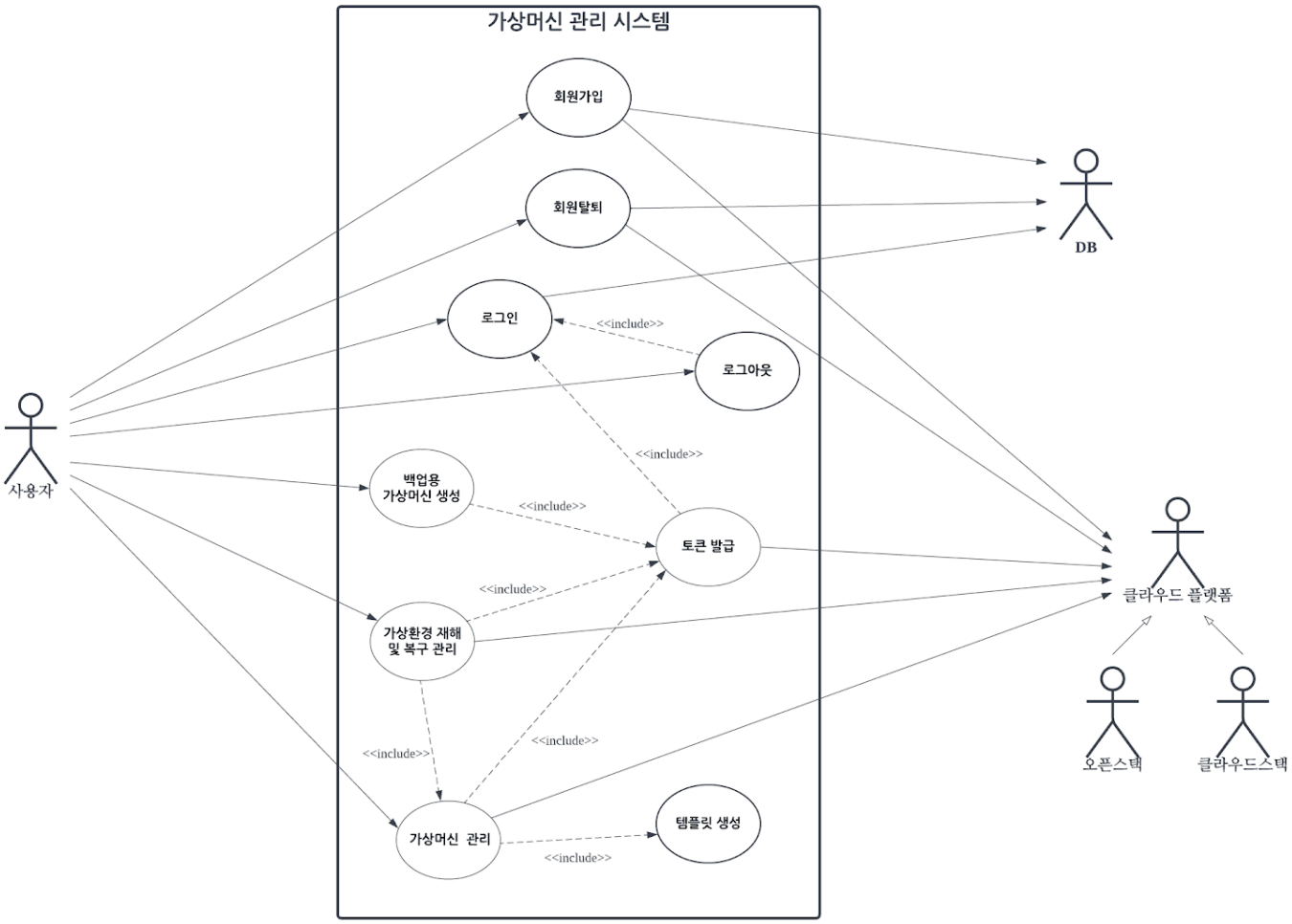

그림 1. Usecase Diagram

3.2. Sequence Diagram

그림 2 에서 그림 10 까지는 유스케이스 기반 시퀀스 다이어그램이다.

- 로그인

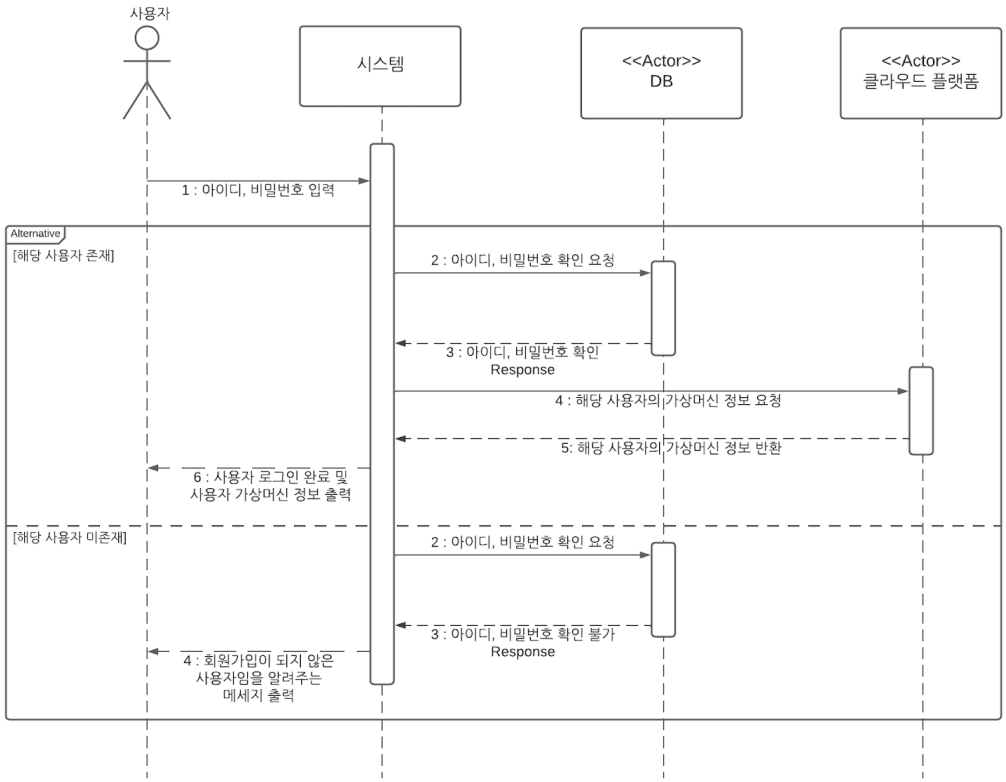

그림 2. Sequence Diagram-로그인

- 로그아웃

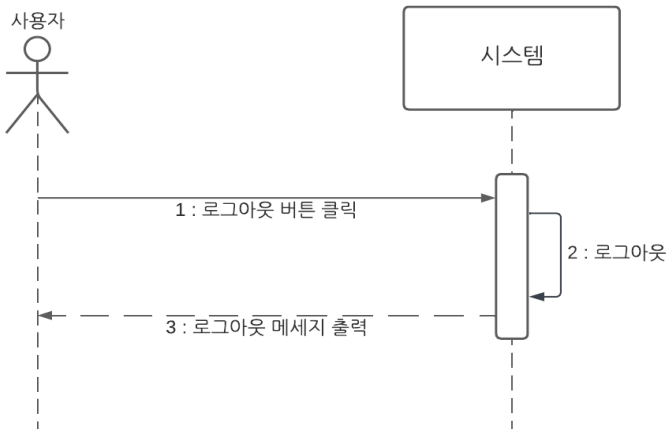

그림 3. Sequence Diagram-로그아웃

- 회원가입

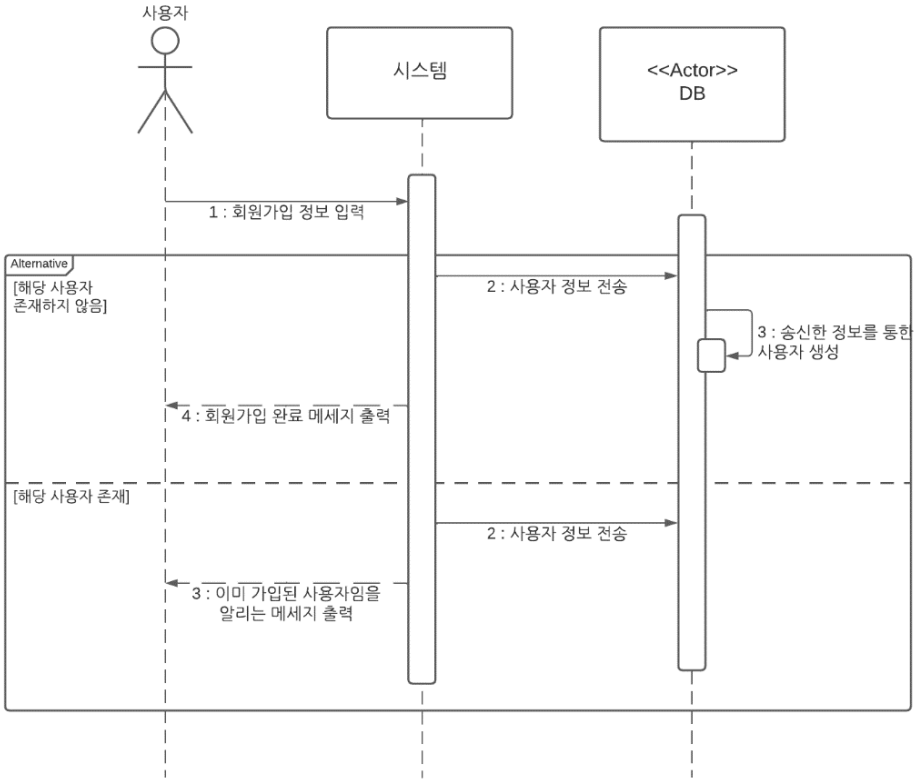

그림 4. Sequence Diagram-회원가입

- 회원탈퇴

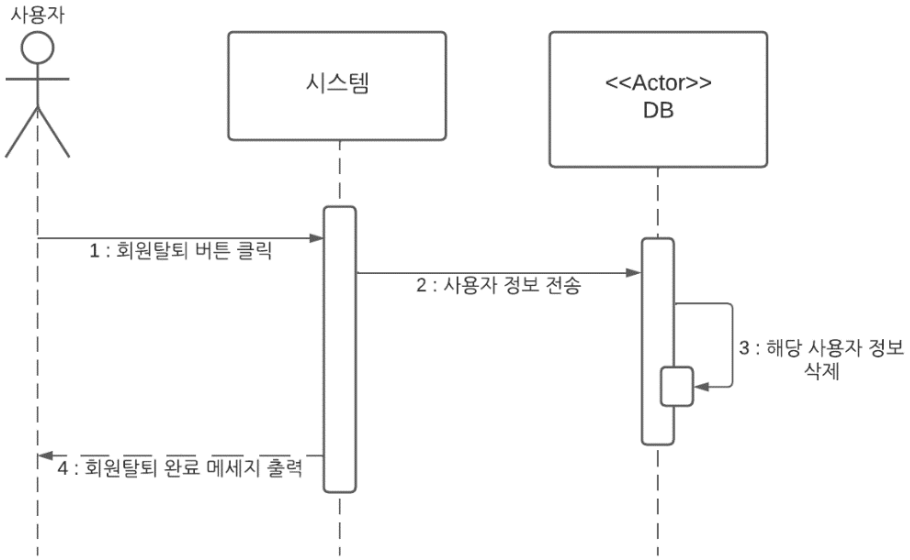

그림 5. Sequence Diagram-회원탈퇴

- 토큰 발급

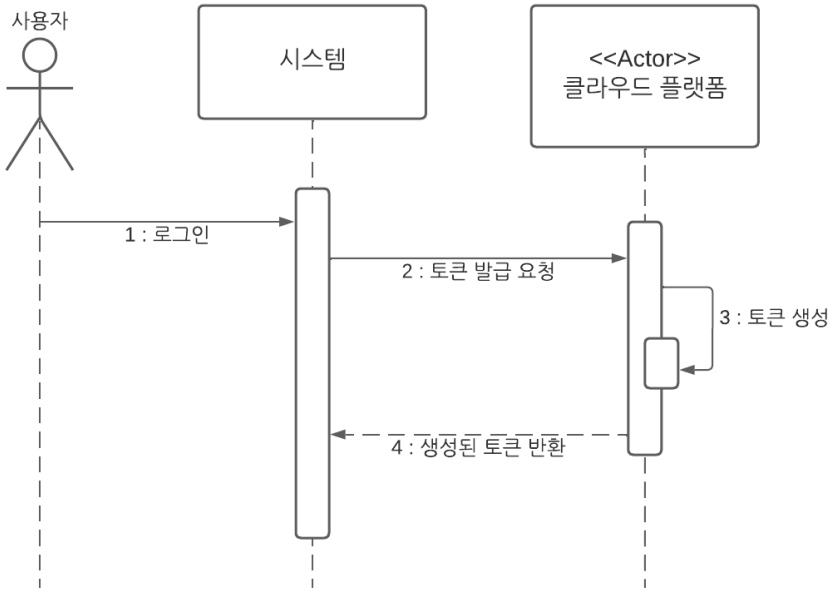

그림 6. Sequence Diagram-토큰 발급

- 템플릿 생성

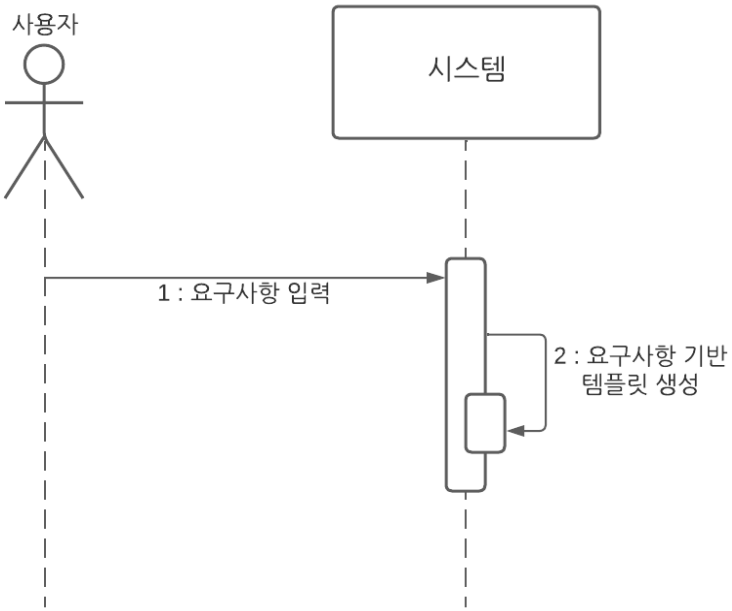

그림 7. Sequence Diagram-템플릿 생성

- 가상머신 관리

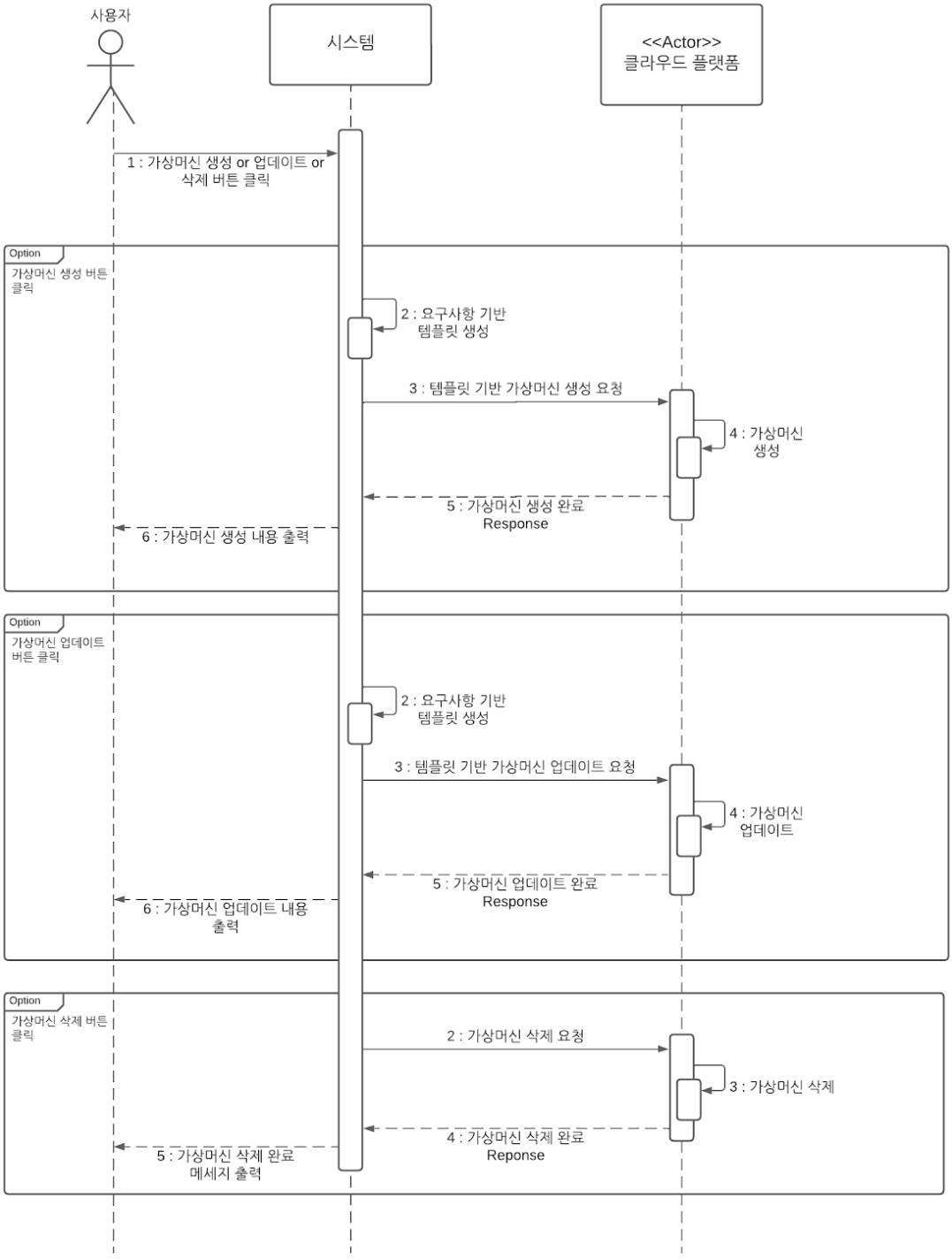

그림 8. Sequence Diagram-가상머신 관리

- 백업용 가상머신 생성

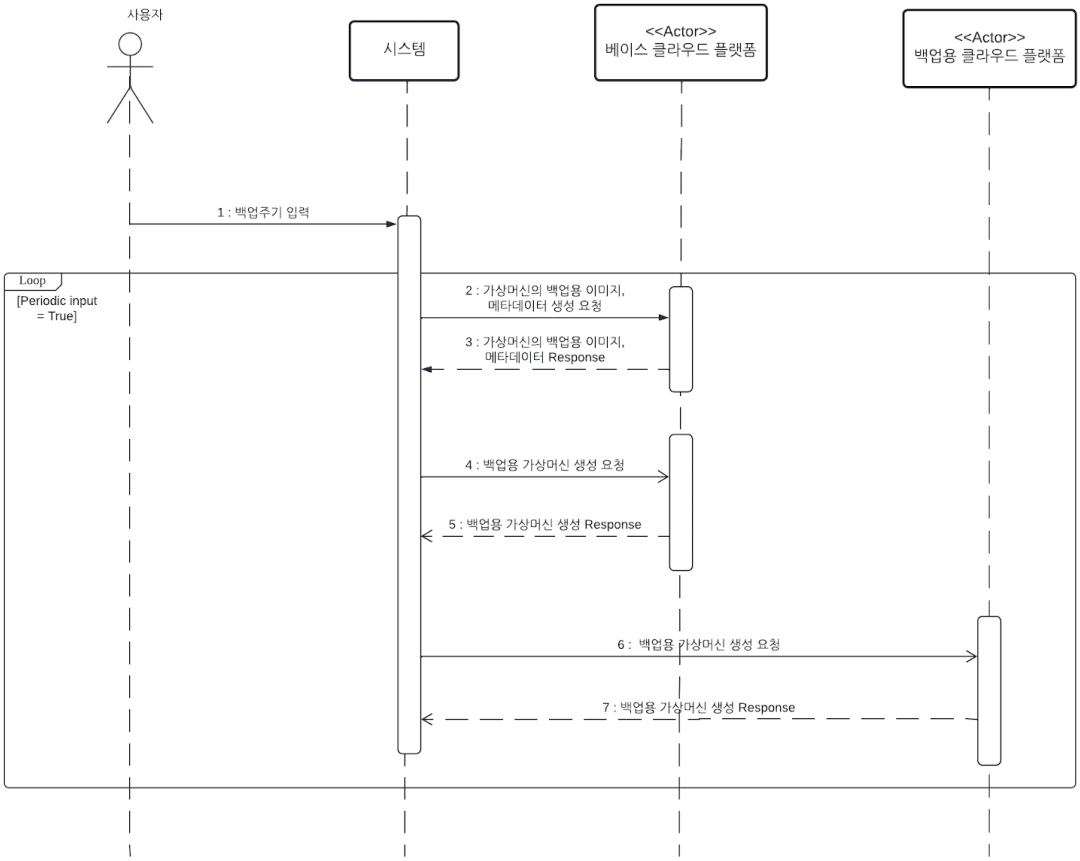

그림 9. Sequence Diagram-백업용 가상머신 생성

- 가상환경 재해 및 복구 관리

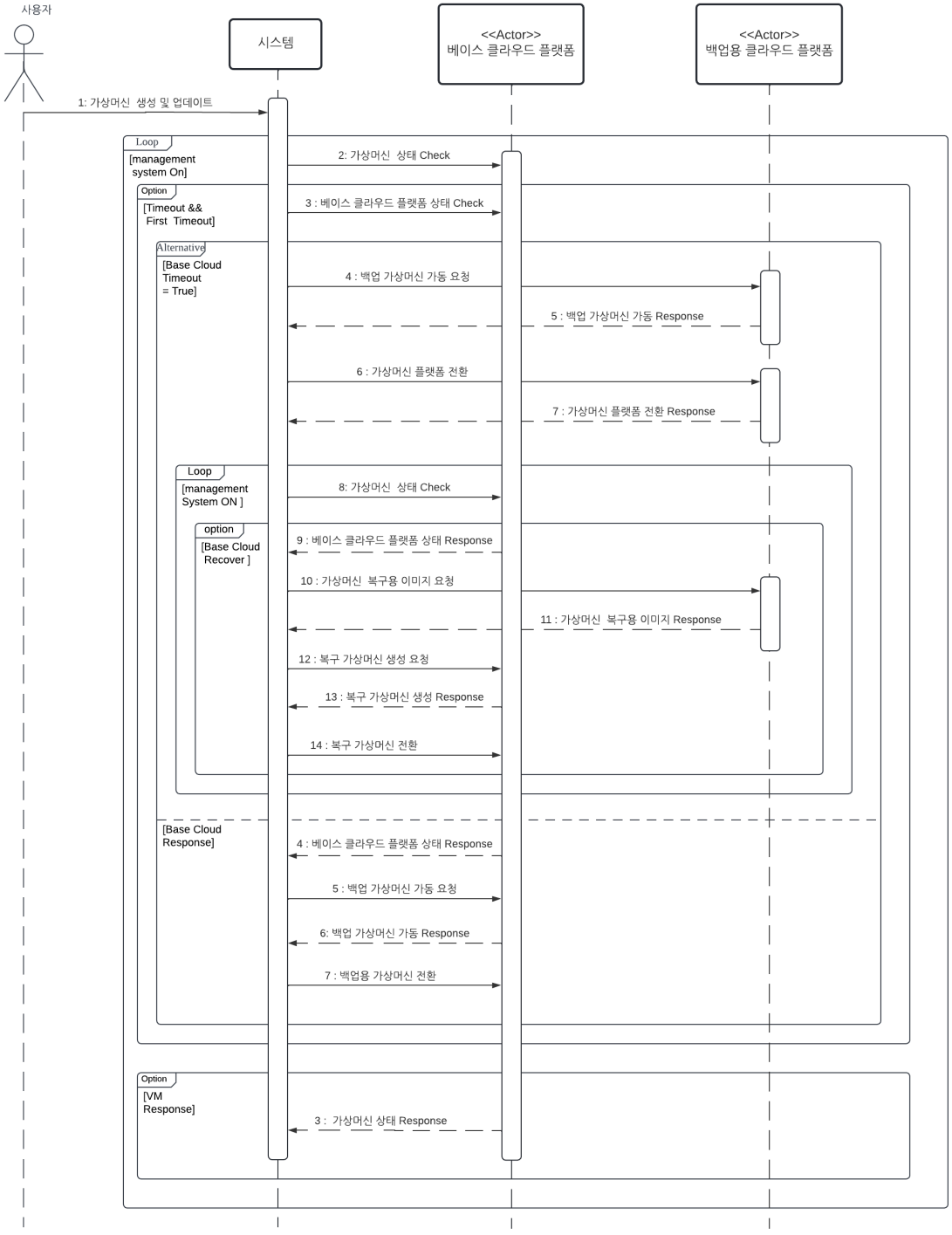

그림 10. Sequence Diagram-가상환경 재해 및 복구 관리

3.3. Deployment Diagram

그림 11은 시스템의 구조를 나타낸 Deployment Diagram이다.

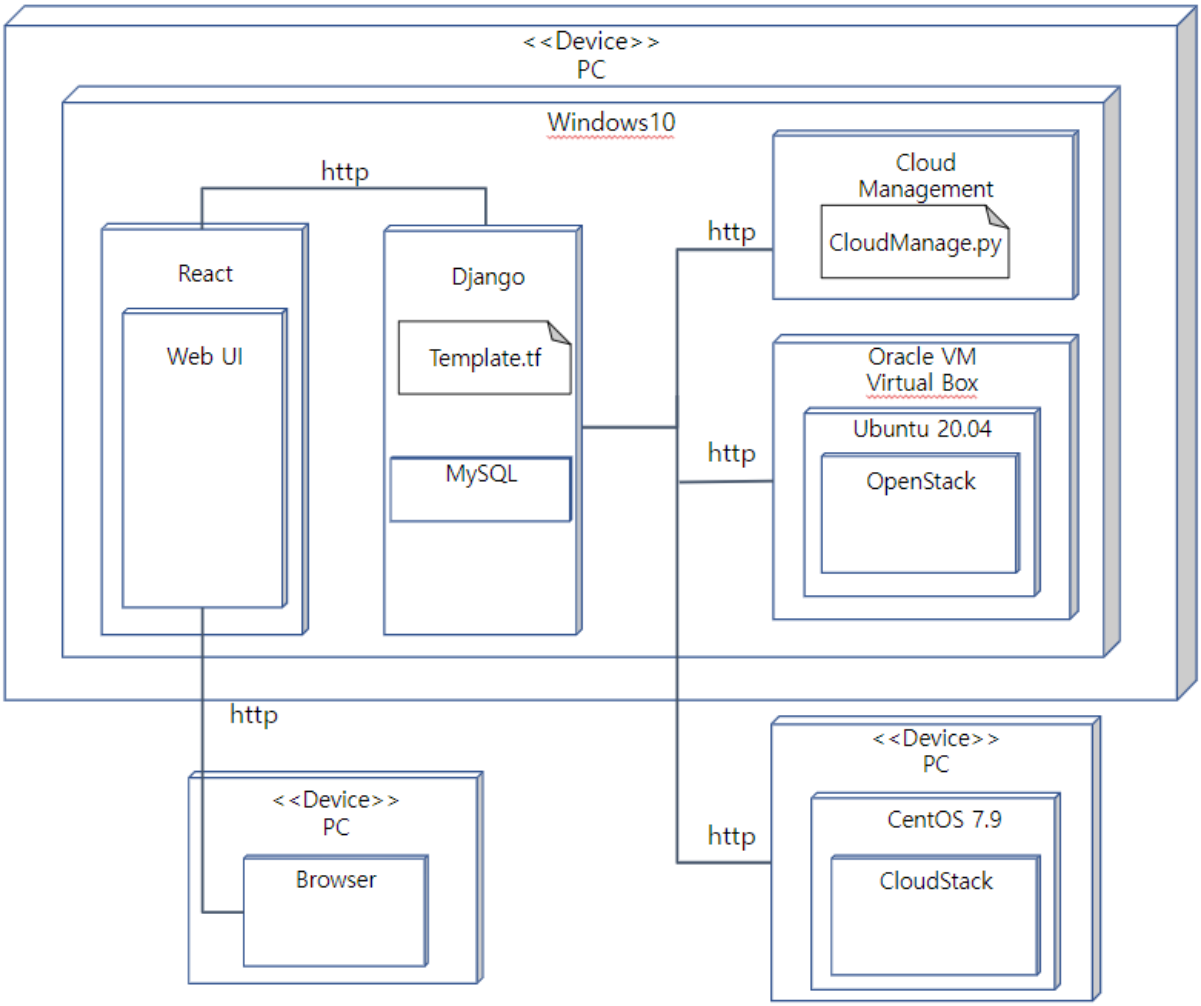

그림 11. Deployment Diagram

시스템은 Web UI를 통해 사용자와 연결된다. 시스템이 Web UI를 통해 사용자의 요구사항 기반 가상머신 생성 또는 업데이트 요청을 받으면 요구사항에 기반한 template.tf 파일을 생성한다. 그 후 시스템은 생성한 파일을 각 클라우드 플랫폼 서버에 전송하여 오케스트레이션을 요청한다. 시스템의 Cloud Management 컴포넌트에서는 생성된 가상머신의 주기적 백업 및 재해 사항 관리를 위해 클라우드 플랫폼과 http통신을 수행한다.

4. 과제 수행 내용 및 중간 결과

4.1. 시스템 환경 구축

4.1.1. 시스템 환경 - 오픈스택

오픈스택 사이드의 환경 구축은 Oracle VM VirtualBox를 이용해 Ubuntu 20.04 버전의 live server 가상머신 상에 수행했다. 우선 오픈스택의 오픈소스를 이용하기 위해 version 3 이상의 python을 가상머신에 설치했다. 그 후 Ubuntu 상의 오픈스택 자동 설치 도구인 devstack을 이용하여 스크립트와 configuration file 설정을 통해 설치를 진행했다. 그림 12는 devstack을 clone하는 스크립트를 보여준다.

```
sudo useradd -s /bin/bash -d /opt/stack -m stack
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
sudo su - stack
git clone https://github.com/openstack-dev/devstack.git -b stable/xena
```

그림 12. devstack 설치를 위한 사용자 추가

devstack을 이용하기 위해 가상머신의 user를 생성하고 해당 user에 Github의 devstack을 clone해준다. 본 과제에서는 오픈스택의 xena 버전을 사용하므로 브랜치를 stable/xena로 설정하여 clone하였다. 그림 13은 오픈스택 configuration을 위한 local.conf 파일의 내용을 보여준다.

```
[[local|localrc]]
HOST_IP=<<HOST_IP>>
ADMIN_PASSWORD=0000
RABBIT_PASSWORD=0000
SERVICE_PASSWORD=0000
DATABASE_PASSWORD=0000

enable_plugin heat https://opendev.org/openstack/heat stable/xena
enable_plugin heat-dashboard https://opendev.org/openstack/heat-dashboard.git stable/xena
enable_service h-eng h-api h-api-cfn h-api-cw heat-dashboard

SWIFT_REPLICAS=1
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
enable_service s-proxy s-object s-container s-account


enable_plugin freezer http://git.openstack.org/openstack/freezer master
enable_plugin freezer-api http://git.openstack.org/openstack/freezer-api.git master
enable_plugin freezer-tempest-plugin http://git.openstack.org/openstack/freezer-tempest-plugin.git master
enable_plugin freezer-web-ui http://git.openstack.org/openstack/freezer-web-ui.git master

export FREEZER_BACKEND='sqlalchemy'
```

그림 13. 오픈스택 configuration 파일

local.conf파일에서는 HOST PC의 IP, 오픈스택 이용을 위한 password, 코어 컴포넌트 외에 사용할 컴포넌트의 버전 및 파라미터를 설정한다. 본 과제에서는 오케스트레이션 컴포넌트 Heat와 백업 데이터 저장 컴포넌트인 Swift, 재해복구 컴포넌트인 Freezer에 대한 설정을 local.conf에 작성했다. configuration file에 내용을 입력한 후 devstack의 stack.sh 파일을 실행하면 오픈스택이 설치된다. 그림 14는 오픈스택의 대시보드를 보여준다.

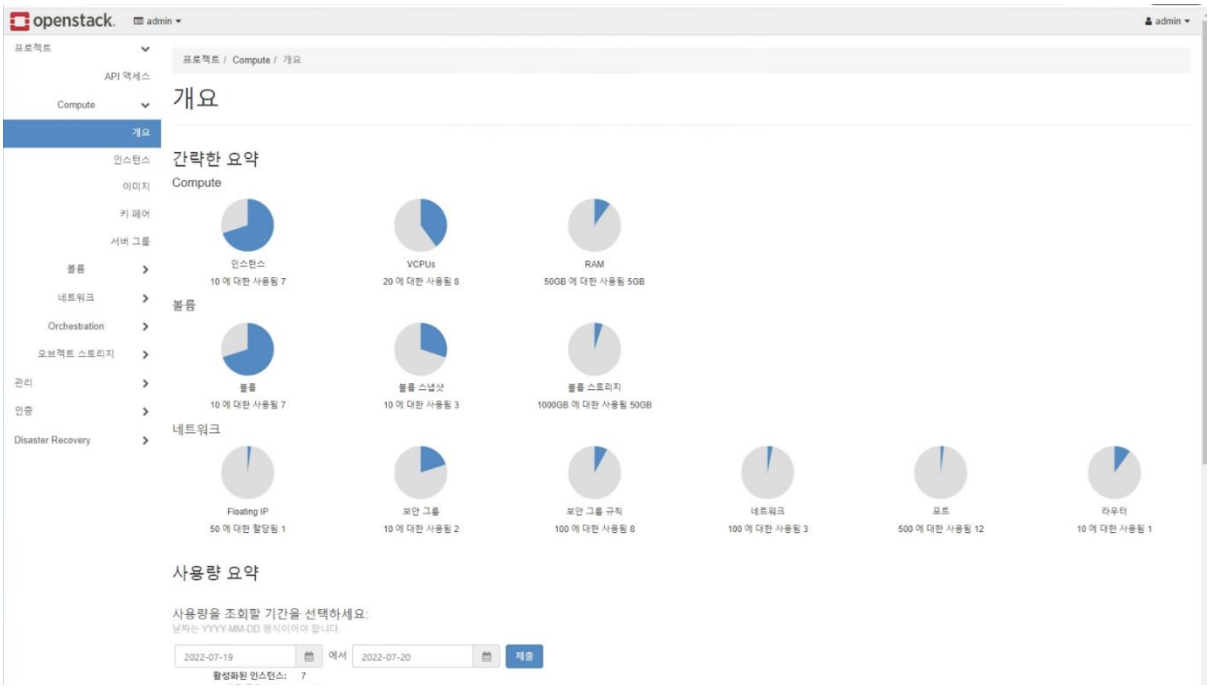

그림 14. 오픈스택 대시보드

오픈스택 대시보드는 웹 주소창에 오픈스택 서버로 사용하는 가상머신의 IP를 입력하면 접근할 수 있다.

4.1.2. 시스템 환경 - 클라우드 스택

클라우드스택 사이드의 환경구축은 PC에 직접 centOS 7.9 버전을 설치하여 진행했다. 우선 서버 PC와 클라우드스택의 네트워킹을 위해 브릿지를 만든다. 그 후 서버 PC의 network-scripts 디렉토리에 ifcfg-cloudbr0 파일을 생성한다. 그림 15는 ifcfg-cloudbr0 파일의 내용이다.

```
DEVICE=cloudbr0
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=static
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
IPADDR= <<HOST_IP>>
GATEWAY= <<GATEWAY_ADDRESS>>
NETMASK=255.255.255.0
DNS1=8.8.8.8
DNS2=8.8.4.4
STP=yes
USERCTL=no
NM_CONTROLLED=no
```

그림 15. 네트워크 브릿지 설정 파일

이후 이 브릿지를 사용할 수 있도록 동일 디렉토리 내에 존재하는 PC의 인터페이스 파일을 수정한다. 그림 16은 PC의 인터페이스 파일을 ifcfg-cloudbr0에 맞추어 수정한 내용이다.

```
TYPE="Ethernet"
BRIDGE=cloudbr0
PROXY_METHOD="none"
BROWSER_ONLY="no"
BOOTPROTO="none"
DEFROUTE="yes"
NAME=<<ex)"ens33">>
DEVICE=<<ex)"ens33">>
ONBOOT="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
UUID=<<ex)"7dbce381-f9c7-4e58-be1b-1560e1563dfc">>
NM_CONTROLLED=no
```

그림 16. 템플릿 생성 화면

이 파일들의 생성과 수정을 통해 클라우드스택의 네트워킹을 사용할 수 있다. 이후 클라우드스택의 설정상 호스트 이름도 설정해주어야 하므로 etc 디렉토리 내 hosts 파일의 내용도 수정한다. 그림 17은 호스트 이름 설정을 위해 hosts 파일을 수정한 것을 보여준다.

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
<<HOST_IP>> <<hostname>>.cloud.priv
```

그림 17. 클라우드 스택 host 설정 파일

이후 시스템의 액세스 권한을 효과적으로 제어하기 위해 SELinux(Security-Enhanced Linux) configuration 파일 설정을 통해 SELinux를 허용해준다. 그 후 모든 클라우드 서버의 시간을 동기화 상태로 유지하기 위해 NTP(Network Time Protocol)를 설치한 후 실행한다. 다음으로 클라우드스택 패키지 저장소를 설정한다. 그림 18은 아파치의 공식 배포 정보를 설정하기 위해 클라우드스택 패키지 저장소를 PC에 설정한 것을 보여준다.

```
[cloudstack]
name=cloudstack
baseurl=http://download.cloudstack.org/centos/$releasever/4.17/
enabled=1
gpgcheck=0
```

그림 18. 클라우드스택 패키지 저장소

아파치의 공식 배포는 바이너리 파일이 아닌 소스코드이기 때문에 커뮤니티에서 제공하는 yum 저장소 정보를 PC에 입력해야 하기 때문에 이러한 설정이 필요하다.

다음으로 NFS(Network File System) 설정을 한다. 이 설정은 Primary/Secondary Storage를 위해서 필요한 설정이다. export 디렉토리 내에 primary storage와 secondary storage 역할을 수행하는 디렉토리를 만든다. 이후 etc 디렉토리 내 idmapd.conf 파일 중 General 섹션의 Domain을 cloud.priv로 수정한다. 동일 디렉토리 내 sysconfig 디렉토리의 nfs 파일에 NFS 포트 설정도 입력한 후, 방화벽 동작을 멈추고 rpcbind와 NFS를 재시작하면 NFS 설정은 마무리가 된다.

Management server 설치를 위해 클라우드스택의 데이터베이스로 동작하는 MySQL을 설치 및 실행한 다음 MySQL Connector 설치를 통해 MySQL을 파이썬과 연결해준다. 그 후 명령어를 통

해 management server를 설치하고 시스템 템플릿을 다운로드한다. 이후의 과정에서 이 템플릿을 이용하여 시스템 가상머신을 생성한다. 이후 하이퍼바이저 설정이 끝나면 클라우드스택 설치가 완료된다. 클라우드스택은 KVM을 기본 하이퍼바이저로 이용하기 때문에 본 과제에서도 하이퍼바이저로 KVM을 운용하기로 했다.

설치가 완료되면 오픈스택과 마찬가지로 웹 주소창에 서버 PC의 IP를 입력하여 대시보드에 접근이 가능하다. 그림 19는 설치 완료 후 클라우드스택의 대시보드를 보여준다.

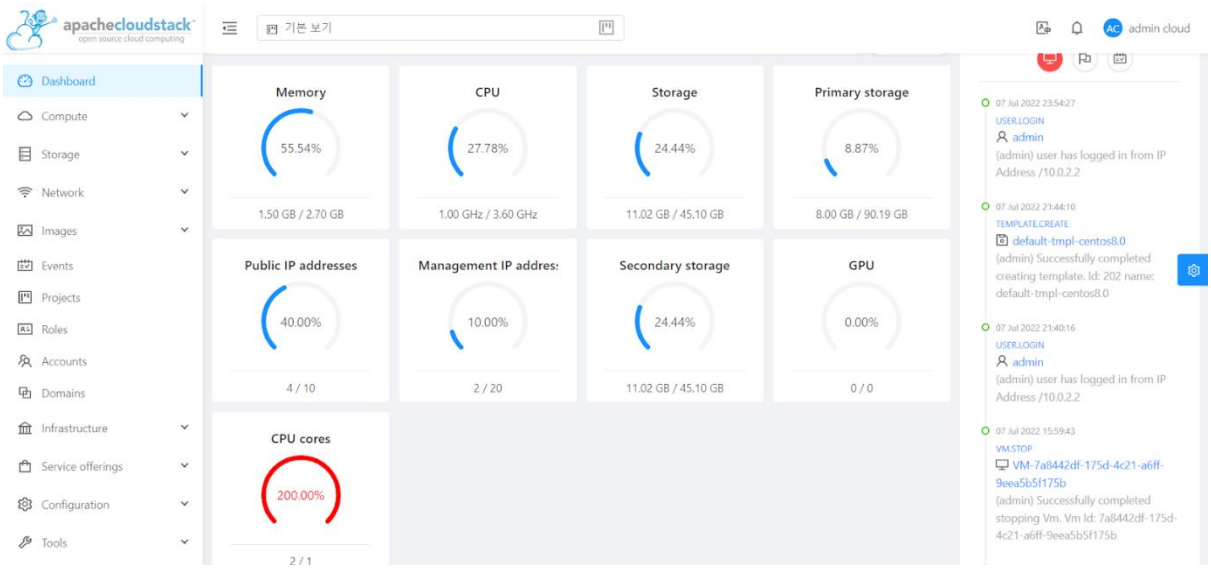

그림 19. 클라우드스택 대시보드

4.2. Freezer 백업 프로세스 분석

4.2.1. Freezer 아키텍처 개요

그림 20은 오픈스택에서 제공하는 재해복구 서비스인 Freezer의 아키텍처를 나타낸 그림이다.

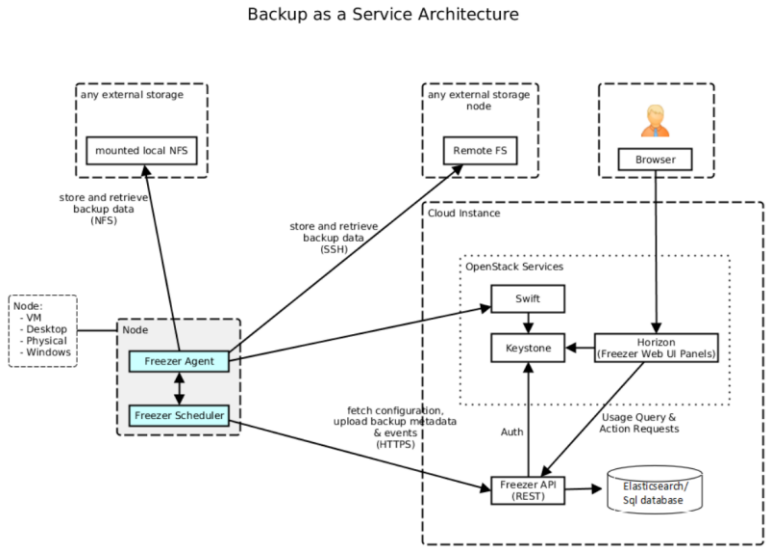

그림 20. Freezer 의 아키텍처

Freezer는 가상머신 백업 외에도 파일시스템 백업, 볼륨 백업 등 다양한 모드의 백업 기능을 지원한다. 또한 백업 시 다양한 압축 알고리즘(gzip, bzip2, xz 등)으로 백업 데이터를 생성할 수 있고 주기적인 백업과 복구가 가능하도록 스케줄 기능을 지원한다.

Freezer는 Freezer API, Freezer Agent, Freezer Scheduler, Freezer Web UI, Elasticsearch로 구성된다. Freezer API는 서버 측에, Agent 및 Scheduler는 실질적으로 백업과 복구 작업을 수행하는 노드 쪽에 설치되어 상호작용을 한다.

사용자가 Horizon 혹은 CLI를 통해 명령을 전달하면 Freezer API는 해당 정보를 Elasticsearch에 저장하고 Freezer-Scheduler에게 관리자 명령을 전달한다. Freezer Agent는 원격지의 파일 시스템 혹은 Swift를 통해 백업된 데이터를 저장한다.

4.2.2. Freezer 아키텍처의 구성요소

- Freezer Web UI

- Freezer API와 상호 작용하는 웹 인터페이스
- 여러 노드에 대한 백업 및 동기화 설정 제공

- Freezer Scheduler(Freezer-client)

- 백업이 실행 될 노드에서 실행 중인 클라이언트
- VM, Physical Node(컨트롤, 컴퓨트 노드) 등에 설치
- 여러가지 작업(백업 및 복구)에 대한 스케줄 관리

- Freezer Agent

- 실행 중인 시스템에서 백업 및 복구에 대한 실질적인 수행
- 데이터를 저장소에 백업 및 복구

- Freezer API

- Job, Session 등과 같은 백업, 복구에 대한 정보를 데이터베이스에 저장 요청
- 여러 작업에 대한 인터페이스 제공(정보 관리)

- Elasticsearch

- 작업 상태 등과 같은 여러 정보를 저장하고 검색하기 위해 Freezer-API에서 사용하는 백엔드 DB

4.2.3. Freezer Agent 명령어를 이용한 가상머신 백업

그림 21은 Freezer Agent 명령어를 나타낸 그림이다.

```
root@controller:~# freezer-agent --backup-name Freezer_backup --mode nova --stor
age swift --container Freezer_Backups --action backup
```

그림 21. Freezer Agent 명령어

가상머신 백업 시에 스케줄링이나 Freezer API 없이 단독으로 Freezer Agent 명령어를 이용해 백업이 가능하다. Freezer Agent 명령어의 mode 옵션으로 Nova, FS, Database 등의 키워드가 명시되어야 한다. 옵션에 명시된 키워드에 따라 가상머신 스냅샷, 대상 경로의 파일 혹은 디렉토리 백업, 데이터베이스 백업 등의 명령이 수행된다.

Freezer Agent 명령어의 action 옵션으로 backup과 restore가 명시되며, 그 외에 container 옵션에는 Swift에 데이터를 저장하기 위한 container의 이름이 명시되어야 한다. 그림 22와 그림 23은 Freezer Agent 명령어와 실행결과를 나타낸 그림이다.

```
+--------------------------------------+-----------+--------+----------------------------------+----
--------------------+---------+
| ID                                   | Name      | Status | Networks                         | Ima
ge                   | Flavor  |
+--------------------------------------+-----------+--------+----------------------------------+----
--------------------+---------+
| f163a790-a4fb-4658-a6ba-aeb487d2f439 | instance1 | ACTIVE | public=172.24.4.7, 2001:db8::1e4 | cir
ros-0.5.2-x86_64-disk | m1.tiny |
+--------------------------------------+-----------+--------+----------------------------------+----
--------------------+---------+
kojunsung@server:~$ freezer-agent --action backup --nova-inst-id f163a790-a4fb-16ba-aeb487d2f439 --s
torage swift --container nova-backup --mode nova --engine nova --no-incremental true --log-file nova
-bkp.log
```

그림 22. Freezer Agent 명령어

```
+---------------------+-----------------------------+
|       Property      |            Value            |
+---------------------+-----------------------------+
|  curr backup level  |              0              |
|     fs real path    |             None            |
|    vol snap path    |             None            |
|      client os      |            linux            |
|    client version   |           10.0.0            |
|      time stamp     |          1657586387         |
|        action       |            backup           |
|     always level    |                             |
|     backup media    |             nova            |
|     backup name     |           nova-bkp          |
|      container      | /home/kojunsung/nova-backup |
|  container segments |                             |
|       dry run       |                             |
|       hostname      |            server           |
|    path to backup   |                             |
|      max level      |                             |
|         mode        |             nova            |
|       log file      |         nova-bkp.log        |
|       storage       |            local            |
|        proxy        |                             |
|     compression     |             gzip            |
|       ssh key       | /home/kojunsung/.ssh/id_rsa |
|     ssh username    |                             |
|       ssh host      |                             |
|       ssh port      |              22             |
| consistency checksum |                             |
+---------------------+-----------------------------+
```

그림 23. Freezer Agent 명령어 실행결과

그림 24 는 백업데이터가 저장된 백업 컨테이너를 나타낸 그림이다.

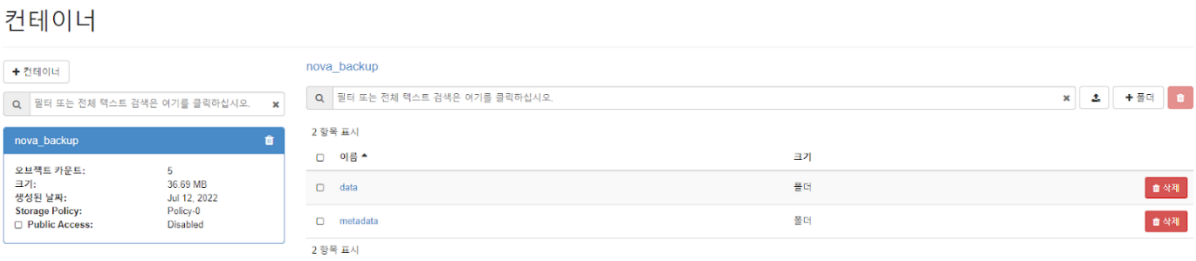

그림 24. 백업 컨테이너 생성 결과

가상머신 백업 결과 백업 컨테이너 안에 data 와 metadata 가 생성된다. Data 에는 가상머신 복구 시에 필요한 백업 데이터가 저장되고 metadata 에는 백업 시 사용한 모드, 백업된 가상머신의 uuid, 이름, 상태, 스냅샷 이미지의 uuid 등 백업과 복구 시점에 필요한 정보가 JSON 포맷으로 저장된다.

4.3. 사용자 맞춤형 가상환경 생성 프로세스 구현

4.3.1. 프로세스 개요

사용자 맞춤형 가상환경 생성 프로세스는 Python과 REST API를 이용하여 구현하였다. 사용자 하드웨어, 소프트웨어 요구사항을 반영한 가상머신 생성을 위해 클라우드 플랫폼의 오케스트레이션 컴포넌트를 활용한다.

현재 가상환경 생성 프로세스는 오픈스택 클라우드 플랫폼에 한하여 구현했으며 추후 Terraform 오픈소스 소프트웨어를 활용하여 오픈스택 플랫폼과 클라우드스택 플랫폼에서 동시에 오케스트레이션이 실행되도록 적용할 계획이다. 그림 25는 사용자 맞춤형 가상환경 생성 프로세스를 나타낸 그림이다.

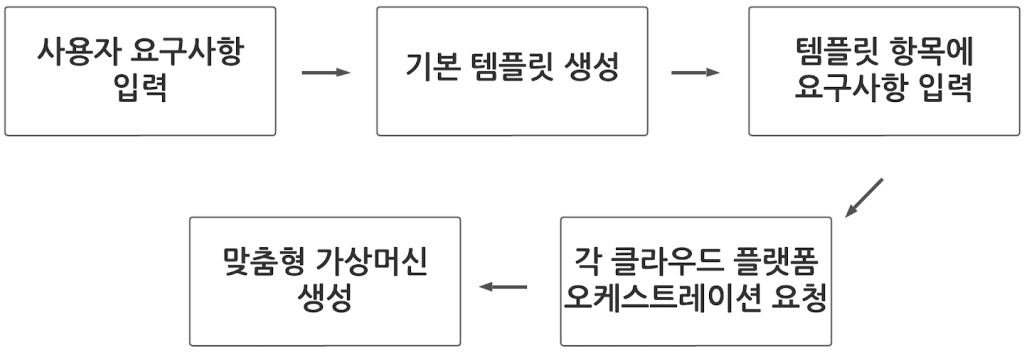

그림 25. 맞춤형 가상환경 생성 프로세스

사용자 요구사항 입력 단계에서는 요구사항 선택지가 출력되고 사용자는 요구사항을 선택한다. 사용자의 요구사항 선택이 완료되면 시스템은 가상머신 생성에 필요한 사항이 입력되어 있는 템플릿 양식을 출력한다.

그 후 시스템은 사용자가 입력한 요구사항을 해당 템플릿 양식에 입력하여 사용자 맞춤형 요구사항 템플릿을 생성한다. 시스템은 REST API를 이용하여 오픈스택 서버에 요구사항 템플릿 기반 오케스트레이션을 요청한다. 그 후 오픈스택 서버가 맞춤형 가상머신을 생성하고 결과를 반환한 후 해당 프로세스는 종료된다.

4.3.2. 기능 구현

맞춤형 가상환경 생성 프로세스는 4.3.1의 프로세스를 바탕으로 구현하였다. 아래 그림 26은 토큰 발급을 위해 작성한 코드이다.

```
def token(self):
    # data2 = json.loads(request.body)
    # Admin으로 Token 발급 Body
    token_payload = {
        "auth": {
            "identity": {
                "methods": [
                    "password"
                ],
                "password": {
                    "user": {
                        "name": "admin",
                        "domain": {
                            "name": "Default"
                        },
                        "password": "0000"
                    }
                }
            }
        }
    }

    # Openstack keystone token 발급
    auth_res = requests.post("http://"+address+"/identity/v3/auth/tokens",
        headers = {'content-type' : 'application/json'},
        data = json.dumps(token_payload))

    #발급받은 token 출력
    admin_token = auth_res.headers["X-Subject-Token"]
    print("token : \n",admin_token)
    return admin_token
```

그림 26. 토큰 발급 함수

그림 26의 token함수는 API를 사용하기 위한 인증 토큰을 발급받는 함수이다. token_payload에 사용자 정보(사용자명, 패스워드)를 입력한다.

다음 token_payload를 바탕으로 토큰 발급 REST API 주소로 토큰을 요청한 후 발급받는다. 아래 그림 27은 토큰 발급 함수를 사용한 코드이다.

```
#API 요청을 위한 인증 토큰 발급
admin_token= self.token()
```

그림 27. 토큰 발급 함수 사용코드

토큰 발급은 그림 27의 token()을 통해 그림 26에서 정의한 토큰 발급 함수를 사용한다. 아래 그림 28은 그림 25의 사용자 요구사항 입력 단계에서 사용자 요구사항을 출력하고 입력 값을 변수로 저장하는 것에 대해 작성한 코드이다.

```
#사용자 요구사항중 OS 입력
osList=['ubuntu.json','centos.json','fedora.json']
system_num=int(input("원하는 OS 번호를 입력: 1.Ubuntu 2.CentOS 3.Fedora \n"))

#사용자 요구사항중 SW 입력
softInfo = ['vim','filezilla','ftp','default-jdk','synaptic']
softNum = list(map(int, input("원하는 소프트웨어 번호를 입력: 1.vim 2.filezilla 3.ftp 4.default-jdk 5.synaptic  입력예시: 1 3 4\n").split()))
```

그림 28. 요구사항 입력 코드

그림 28에서 사용자에게 OS, SW 선택지를 출력하고 사용자는 요구사항을 입력한다. system_num과 softNum 변수는 템플릿에 입력할 요구사항을 정의한다. 아래 그림 29는 그림25의 기본 템플릿 생성 단계와 템플릿 항목에 요구사항 입력 단계에 대해 작성한 코드이다.

```
#기본 템플릿 생성
with open(osList[system_num-1], 'r') as f:
    json_data = json.load(f)

#기본 템플릿 항목에 요구사항 입력
for i in softNum:
    json_data['template']['resources']['myconfig']['properties']['cloud_config']['packages'].append(softInfo[i - 1])
print(json_data['template']['resources']['myconfig']['properties']['cloud_config']['packages'])
```

그림 29. 템플릿 입력 코드

사용자는 그림 29에서 가상머신 생성에 기초적인 사항이 입력되어 있는 템플릿 양식을 불러와 json_data 변수로 할당한다. 다음 시스템은 사용자가 입력한 요구사항을 json_data 변수의 항목에 입력하여 템플릿을 완성한다. 아래 그림 30은 사용자 요구사항이 적용된 JSON 포맷 템플릿의 예시이다.


```
"stack_name": "VM_Orchetration_test",
"template": {
    "heat_template_version": "2018-08-31",
    "description": "This template demonstrates the different ways configuration resources can be used to specify boot-time cloud-init configuration.",
    "resources": {
        "mybox": {
            "type": "OS::Nova::Server",
            "properties": {
            "name": "VM_of_Orchestration_test",
            "flavor": "ds512M",
            "image": "ubuntu",
            "key_name": {
                "get_resource": "demo_key"
            },
            "networks": [
                {
                "port": {
                    "get_resource": "mybox_management_port"
                }
                }
            ],
            "user_data": {
                "get_resource": "myconfig"
            },
            "user_data_format": "RAW"
            }
        },
        "myconfig": {
            "type": "OS::Heat::CloudConfig",
            "properties": {
            "cloud_config": {
                "packages": [
                "vim"
                ]
            }
            }
        },
```


그림 30. 오케스트레이션 템플릿 예제

그림 30의 “myconfig”-”cloud_config”-”package” 항목에서 사용자가 선택한 vim 패키지가 입력된 것을 확인할 수 있다. 아래 그림 31는 그림 25의 각 클라우드 플랫폼에 오케스트레이션을 요청하는 단계에 대해 작성한 코드이다.

```
#클라우드 플랫폼 오케스트레이션 요청
user_res = requests.post("http://"+address+"/heat-api/v1/6afe05fbd2cb47a6b149ee3541fb47a6/stacks",
    headers = {'X-Auth-Token' : admin_token},
    data = json.dumps(json_data))
#맞춤형 가상머신 생성 결과 출력
print("stack생성 ",user_res)
```

그림 31. 오케스트레이션 요청 코드

그림 31의 request에서는 그림 29처럼 완성된 템플릿을 입력 값으로 오케스트레이션을 오픈스택 서버에 요청한다. 다음 user_res변수를 출력하여 해당 요청에 따른 결과를 확인한다. 아래 그림 32는 그림 25의 가상환경 생성 단계를 나타낸 그림으로 오른쪽, 왼쪽 그림은 각각 오케스트레이션을 실행한 결과 구축된 환경 토폴로지와 가상머신 생성 콘솔창이다.

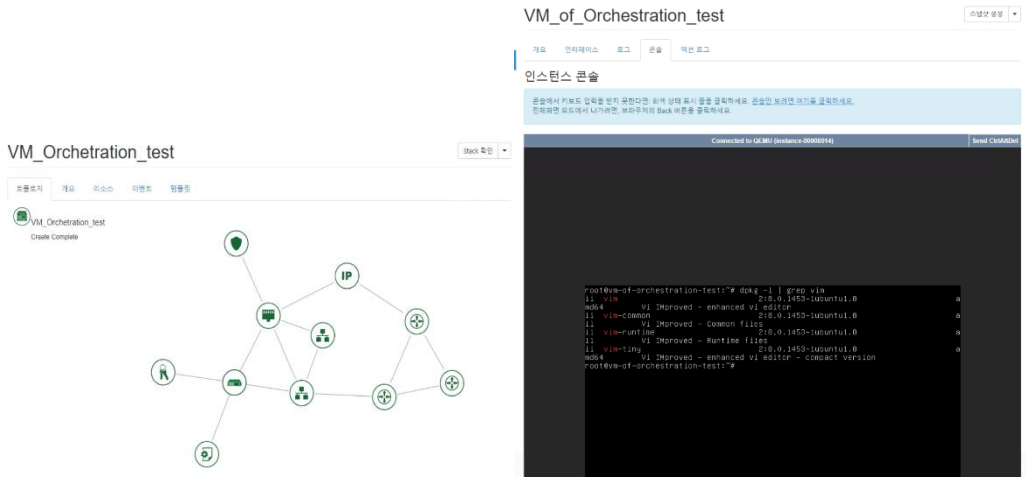

그림 32. 구축 결과

그림 32의 환경 토폴로지에서 사용자가 입력한 SW, 네트워크 설정, 가상머신이 구축된 것을 확인할 수 있다. 또한 아래의 가상머신 콘솔에서 사용자 요구사항에서 입력한 SW(vim)가 설치된 것을 확인할 수 있다.

4.4. 백업 프로세스 구현

4.4.1. 프로세스 개요

백업 프로세스는 오픈스택과 클라우드스택 두 플랫폼을 이용하여 구현하였으며 두 가지의 상황을 고려한 구현을 계획하고 있다. 첫 번째 상황은 사이버 공격이나 소프트웨어 업데이트 중 실패 등의 재해로 인해 오픈스택 클라우드 플랫폼 상에서 사용자의 가상머신 상에 장애가 발생한 경우이다. 그림 33은 첫 번째 상황에 대한 백업 프로세스를 나타낸 그림이다.

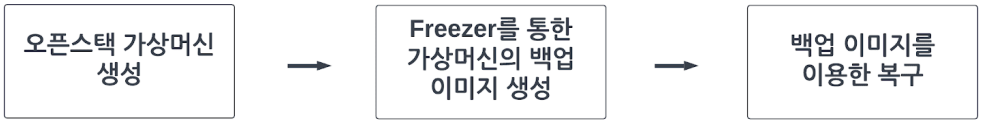

그림 33. 베이스 클라우드 플랫폼의 백업 프로세스

이 상황에서는 Freezer 컴포넌트를 통해 백업해뒀던 이미지로 재해가 발생한 가상머신의 복구를 수행한다. 우선 시스템은 생성된 가상머신에 대하여 Freezer 컴포넌트를 이용하여 주기적으로 백업 이미지를 생성한다. 사용자의 가상머신에 장애가 발생한 경우, 시스템은 최신의 백업 이미지를 이용하여 복구를 진행한다.

두 번째 상황은 정전, 서버 컴퓨터 문제 등의 재해로 인해 오픈스택 클라우드 플랫폼의 서버에 장애가 생겼을 때이다. 그림 34는 두 번째 상황에 대한 백업 프로세스를 나타낸 그림이다.

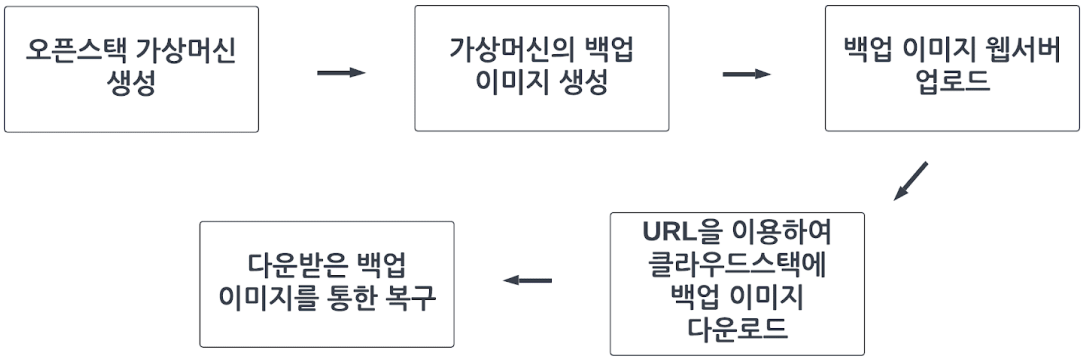

그림 34. 백업용 클라우드 플랫폼의 백업 프로세스

이 상황에서는 Freezer 컴포넌트가 아닌 본 시스템의 주기적 백업 이미지 생성 기능을 통한 복구를 진행한다. 오픈스택 클라우드 플랫폼의 서버 상에 재해가 발생하면 시스템은 웹서버에 업로드 해놓은 백업 이미지 중 최신의 이미지의 URL을 이용하여 클라우드스택에 이미지를 다운받는다. 그 후 다운받은 이미지를 통해 시스템은 클라우드스택 상에서 재해가 발생한 가상머신의 복구를 진행한다.

4.4.2. 백업 프로세스 기능 구현

아래 그림 35는 그림 34의 가상머신의 백업 이미지 생성 단계에서 백업 이미지를 주기적으로 생성하는 기능에 대해 작성한 코드이다.

```
#인스턴스로부터 스냅샷 이미지 생성
def create_img_from_server(self,instacne_name,image_name):
    admin_token= self.token()

    instance_uuid=requests.get("http://"+address+"/compute/v2.1/servers?"+instacne_name,
        headers = {'X-Auth-Token' : admin_token}
        ).json()["servers"][0]["id"]
    print("instance uuid is : \n",instance_uuid)
    openstack_img_payload = {
            "createImage" : {
                "name" : image_name
            }
    }
    #인스턴스 바탕으로 이미지 생성
    user_res = requests.post("http://"+address+"/compute/v2.1/servers/"+instance_uuid+"/action",
        headers = {'X-Auth-Token' : admin_token},
        data = json.dumps(openstack_img_payload))

    print("인스턴스로부터 이미지 생성 ",user_res)
```

그림 35. 백업 이미지 생성코드

시스템은 오픈스택 서버에 백업 이미지를 생성할 가상머신의 uuid를 요청한다. 반환받은 가상머신의 uuid를 통해 이미지 생성을 요청한다. 그림 36은 백업 클라우드 플랫폼인 클라우드스택의 템플릿(이미지) 다운로드 UI를 보여준다.

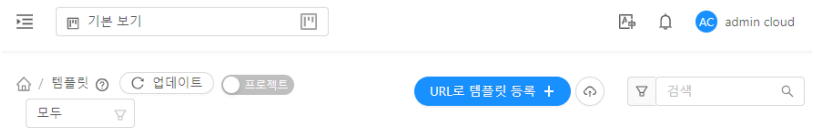

그림 36. 템플릿 생성 버튼

클라우드스택의 템플릿 추가 방법으로는 URL을 통해 추가하는 방법이 있다. 그림 37은 그림 34의 URL을 이용하여 클라우드 스택에 백업 이미지를 다운받는 단계를 나타낸 화면이다.

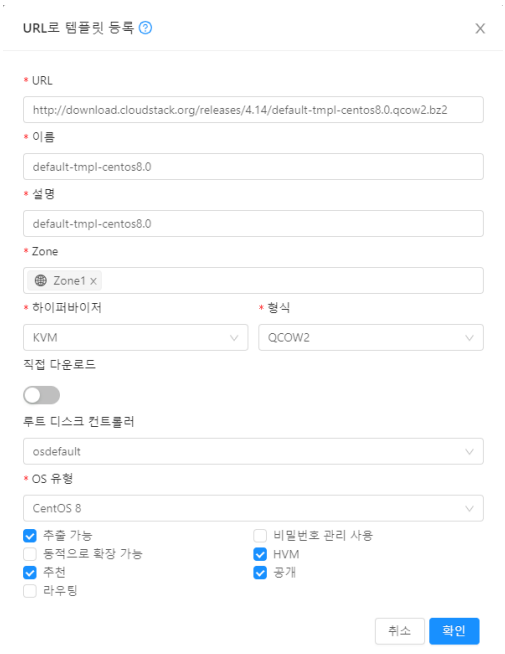

그림 37. 템플릿 생성 화면

URL을 통해 템플릿을 다운로드하는 방법의 필수 파라미터로는 URL, 템플릿 이름, 설명, Zone 설정, 하이퍼바이저 설정, 형식, OS 유형이 있다. 이 파라미터들을 자신이 다운로드하려는 템플릿의 내용에 맞춰 설정해준 후 다운로드를 진행한다. 그림 38은 URL통해 생성한 centos 8 템플릿을 나타낸다.

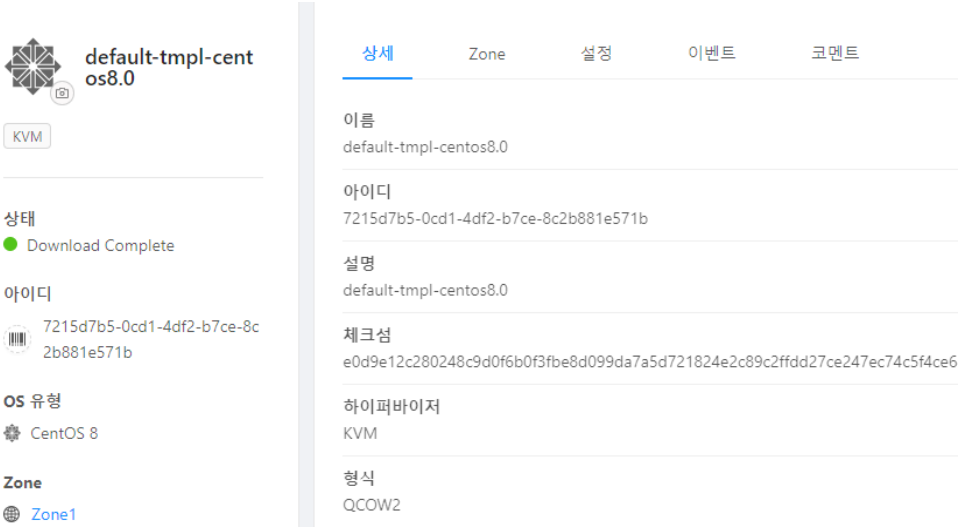

그림 38. 템플릿 생성 결과

URL을 통해 다운로드가 완료된 템플릿은 상태가 'Download Complete'으로 변경된다. 그림 39은 그림 34의 다운받은 백업 이미지를 통한 복구 단계에서 템플릿을 바탕으로 가상머신을 생성하는 화면이다.

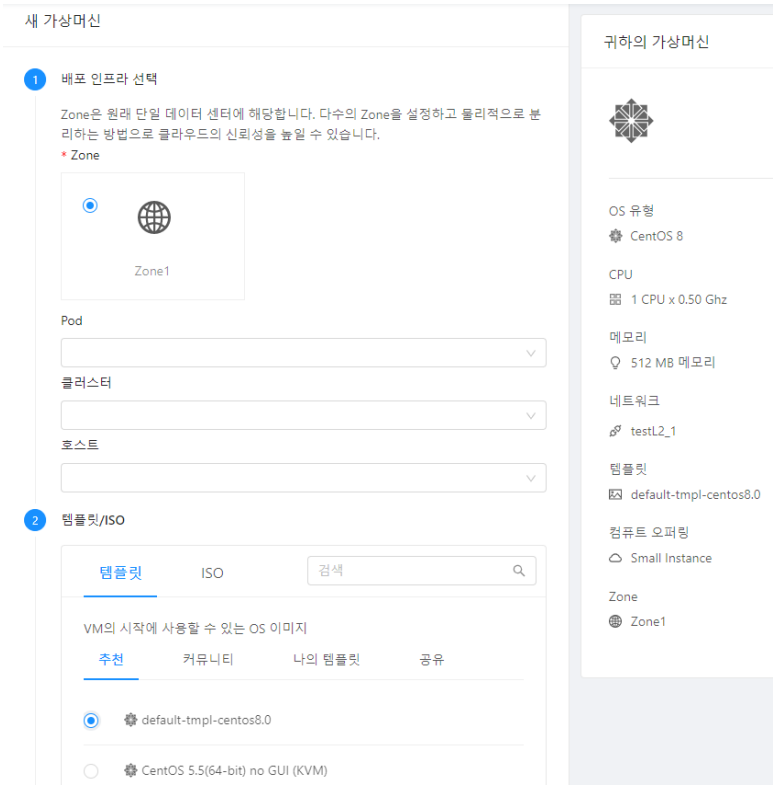

그림 39. 가상머신 생성 화면

가상머신 생성 시에는 다운로드된 템플릿들 중 하나를 고를 수 있다. 본 시스템은 위에서 설명한 주기적 백업 이미지 생성을 통해 생성한 이미지를 웹서버에 저장, 저장된 이미지의 URL을 통해 클라우드스택으로의 다운로드 후 다운로드된 이미지를 통해 백업 가상머신을 생성하는 과정을 구현하는 것을 목표로 개발을 진행중이다.

5. 과제 추진 계획

5.1. 구성원 별 진척도

표 12. 구성원 별 진척도

이름	진척도
공통	필요 지식 습득
이봉훈	사용자 요구사항 기반 템플릿 생성 기능 구현, 백업 이미지 생성 프로세스 구현
고준성	Freezer 서비스 환경구축 및 테스트, Freezer-Agent 기반 가상머신 백업 테스트
김영후	Freezer 코드 디버깅, 웹서버 및 가상머신 생성 및 관리 컴포넌트 설계

5.2. 과제 추진 계획

표 13. 과제 추진 계획

기간 / 수행내용	7월		8월				9월				
	4	5	1	2	3	4	1	2	3	4	5
재해발생 시 알림 기능 구현	이봉훈										
멀티 클라우드 플랫폼 이용 재해복구 기능 구현	고준성										
변경된 요구사항 가상환경에 적용 기능 구현	김영후										
DB 구축 및 사용자 가상환경 정보반환 기능 구현			고준성								
맞춤형 가상환경 Template 생성 컴포넌트 구현			이봉훈								
웹 기반 대시보드 컴포넌트 구현	김영후										
테스트 및 보완						All					
최종보고서 작성									All		