(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2022년09월14일
(11) 등록번호 10-2443172
(24) 등록일자 2022년09월07일

(51) 국제특허분류(Int. Cl.)
G06F 9/50 (2018.01) *G06F 11/14* (2006.01)
G06F 9/44 (2018.01) *G06F 9/445* (2018.01)
(52) CPC특허분류
G06F 9/5077 (2013.01)
G06F 11/14 (2013.01)
(21) 출원번호 10-2017-7009297
(22) 출원일자(국제) 2015년09월24일
심사청구일자 2020년09월17일
(85) 번역문제출일자 2017년04월05일
(65) 공개번호 10-2017-0058955
(43) 공개일자 2017년05월29일
(86) 국제출원번호 PCT/US2015/052057
(87) 국제공개번호 WO 2016/049376
국제공개일자 2016년03월31일
(30) 우선권주장
62/054,903 2014년09월24일 미국(US)
(56) 선행기술조사문헌
US20130326506 A1
(뒷면에 계속)

(73) 특허권자
오라클 인터내셔날 코포레이션
미국, 캘리포니아 94065, 레드우드 쇼어스 엠에스 5오피7, 오라클 파크웨이 500
(72) 발명자
이슬람 나즈룰
미국 95054 캘리포니아 산타 클라라 아파트먼트 202 비스타 클럽 서클 1555
린드홀름 자콥
미국 01803 매사추세츠 벌링톤 네트워크 드라이브 45
(뒷면에 계속)
(74) 대리인
박장원

전체 청구항 수 : 총 19 항 심사관 : 유진태

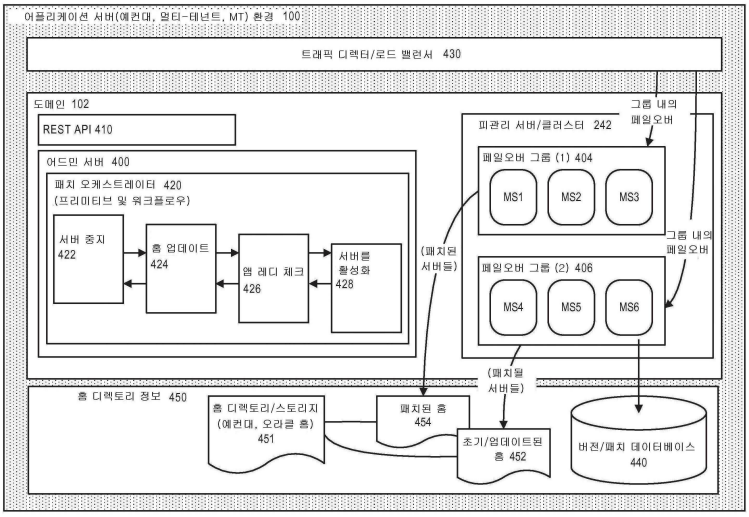
(54) 발명의 명칭 **멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템 및 방법**

(57) 요 약

하나의 실시예에 따르면, 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템 및 방법이 본 명세서에 기술된다. 상기 시스템은 테넌트에 의한 사용을 위해 상기 테넌트와 하나 이상의 파티션들을 관련시킬 수 있고, 파티션은 도메인의 런타임 및 어드미니스트레이티브 서브디비전(administrative subdivision) 또는 슬라이스

(뒷면에 계속)

대 표 도



(slice)이다. 패치 프로세스는 어플리케이션 서버 클러스터링 환경에 의해 제공되는 고 가용성 특징들의 장점을 취하여, 인터럽션(interruption)이 없이 또는 다운타임이 영(zoro)인 채로 동작할 수 있게 하는 도메인의 능력을 유지하는 제어된 롤링 재시작에서 패치를 적용할 수 있다. 이 프로세스는 패치되지 않은 또는 이전 버전의 어플리케이션 서버, 어플리케이션 또는 가능한 롤백을 위한 다른 소프트웨어 컴포넌트를 보존하는 것 또는 복구 불가능한 에러의 이벤트 시 자동 리버전을 제공하는 것을 포함하는 복잡한 또는 장기간 실행되는 태스크들을 자동화하기 위해 이용될 수 있다.

(52) CPC특허분류
G06F 8/65 (2013.01)
G06F 8/658 (2018.02)

(72) 발명자
도르 조슈아
미국 95119 캘리포니아 산호세 큐리 드라이브 341
카쏘 크리스토퍼 에스.
미국 94022 캘리포니아 로스 알토스 마빈 애비뉴 90
발라수브람만얌 야미니 케이.
인도 560085 방갈로 바나샨카리 제3 스테이지 바나기리나가 하우징 레이아웃 브하바니 제6 블록 아딧야 닐라야 #32
리우 스티븐
중국 100020 베이징 챠오 양 멘 웨이 스트리트 넘버 16 차이나 라이프 타워 1002
모르다니 라지브
미국 94089 캘리포니아 서니배일 린즈 테라스 1038
쿰마 아브히짓
미국 95014 캘리포니아 쿠퍼티노 요크셔 드라이브 1125

(56) 선행기술조사문헌
US20120254445 A1
US20120324069 A1
US20120147894 A1
US20130086235 A1
US20110265168 A1
JP2005092803 A*
*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템으로서,

상기 시스템은 하나 이상의 컴퓨터들을 포함하고, 상기 하나 이상의 컴퓨터들은, 복수의 파티션들을 지원하는 도메인의 일부로서 상기 하나 이상의 컴퓨터들 상에서 실행되는 어플리케이션 서버 환경 내에서 작동하는 복수의 피관리(managed) 서버 인스턴스들을 포함하며, 상기 하나 이상의 컴퓨터들은,

상기 어플리케이션 서버 환경 내에서 이용될 수 있는 복수의 전개가능한 리소스들(deployable resources),

도메인 내에서 전개가능한 리소스들의 그룹들을 정의하는 하나 이상의 리소스 그룹 템플릿들, 그리고

하나 이상의 파티션들을 포함하며, 각각의 파티션은 상기 도메인의 어드미니스트레이티브 및 런타임 서브디비전(administrative and runtime subdivision)을 제공하며, 상기 시스템은 테넌트(tenant)에 의한 이용을 위해, 하나 이상의 파티션들 및 리소스 그룹들을 상기 테넌트와 관련시킬 수 있고,

상기 시스템은 인터럽션(interruption)이 없이 상기 도메인의 오퍼레이션을 유지하는 제어된 방식으로 상기 복수의 피관리 서버 인스턴스들을 업데이트하도록 패치들을 적용하기 위해 어플리케이션 서버 환경에 구축된 가용성 특징들(availability features)의 장점을 취하는 패치 특징을 포함하며, 상기 시스템은,

상기 복수의 피관리 서버 인스턴스들에 대해 패치된 디렉토리로서 패치의 롤아웃(rollout)을 수행하는 동작,

각각의 피관리 서버 인스턴스에 대해, 피관리 서버 인스턴스를 셧-다운하고, 상기 패치된 디렉토리의 인스턴스를 사용하도록 상기 피관리 서버 인스턴스를 업데이트하고, 상기 피관리 서버 인스턴스를 업데이트하도록 상기 피관리 서버 인스턴스를 다시 시작하는 동작;

상기 복수의 피관리 서버 인스턴스들에 대한 패치의 롤아웃 동안, 특정 세션과 관련된 요청을 수신하면, 상기 요청을 프로세싱하기 위해 상기 복수의 피관리 서버 인스턴스들 중 제1 피관리 서버 인스턴스에서 상기 특정 세션을 로드하려고 시도하는 동작; 그리고

상기 복수의 피관리 서버 인스턴스들 중 제1 피관리 서버 인스턴스가 상기 특정 세션을 로드할 수 없는 경우, 상기 특정 세션을 로드할 수 있는 상기 복수의 피관리 서버 인스턴스들 중 다른 피관리 서버 인스턴스의 표시를 제공하고, 상기 요청을 프로세싱하기 위해, 상기 특정 세션이 상기 복수의 피관리 서버 인스턴스들 중 제2 피관리 서버 인스턴스에서 로드되도록 지시하는 동작을 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템.

청구항 2

제1항에 있어서,

패치 프로세스의 모니터링 및 에러 처리를 제공하는 오케스트레이션 프레임워크(orchestration framework)를 더 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템.

청구항 3

제1항에 있어서,

상기 어플리케이션 서버 환경은 Java EE 어플리케이션 서버를 포함하고, 각각의 리소스 그룹 템플릿은, 하나 이상의 관련 어플리케이션들 및 상기 하나 이상의 관련 어플리케이션들이 의존하는 리소스들을 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템.

청구항 4

제1항에 있어서,

상기 멀티테넌트 어플리케이션 서버 환경 내에서 패치를 지원하는 노드 관리자를 더 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템.

청구항 5

제1항에 있어서,

상기 시스템은 클러스터화된 환경에서 노드들에 패치들을 적용하기 위해 이용되는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템.

청구항 6

제1항에 있어서,

상기 시스템은, 클라우드 환경 내에서 동작하는 복수의 테넌트들을 지원하기 위해 상기 클라우드 환경 내에서 제공되는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템.

청구항 7

멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법으로서,

하나 이상의 컴퓨터들에서, 복수의 파티션들을 지원하는 도메인의 일부로서 상기 하나 이상의 컴퓨터들 상에서 실행되는 어플리케이션 서버 환경 내에서 작동하는 복수의 피관리(managed) 서버 인스턴스들을 비롯하여, 상기 어플리케이션 서버 환경 내에서 이용될 수 있는 복수의 전개가능한 리소스들과, 도메인 내에서 전개가능한 리소스들의 그룹들을 정의하는 하나 이상의 리소스 그룹 템플릿들과, 그리고 하나 이상의 파티션들을 제공하는 단계 - 각각의 파티션은 상기 도메인의 어드미니스트레이티브 및 런타임 서브디비전을 제공함 - 와; 그리고

테넌트(tenant)에 의한 이용을 위해, 하나 이상의 파티션들 및 리소스 그룹들을 상기 테넌트와 관련시키는 단계를 포함하고,

시스템은 인터럽션이 없이 상기 도메인의 오퍼레이션을 유지하는 제어된 방식으로 상기 복수의 피관리 서버 인스턴스들을 업데이트하도록 패치들을 적용하기 위해 어플리케이션 서버 환경에 구축된 가용성 특징들(availability features)의 장점을 취하는 패치 특징을 포함하며, 상기 시스템은,

상기 복수의 피관리 서버 인스턴스들에 대해 패치된 디렉토리로서 패치의 롤아웃(rollout)을 수행하는 동작,

각각의 피관리 서버 인스턴스에 대해, 피관리 서버 인스턴스를 셧-다운하고, 상기 패치된 디렉토리의 인스턴스를 사용하도록 상기 피관리 서버 인스턴스를 업데이트하고, 상기 피관리 서버 인스턴스를 업데이트하도록 상기 피관리 서버 인스턴스를 다시 시작하는 동작;

상기 복수의 피관리 서버 인스턴스들에 대한 패치의 롤아웃 동안, 특정 세션과 관련된 요청을 수신하면, 상기 요청을 프로세싱하기 위해 상기 복수의 피관리 서버 인스턴스들 중 제1 피관리 서버 인스턴스에서 상기 특정 세션을 로드하려고 시도하는 동작; 그리고

상기 복수의 피관리 서버 인스턴스들 중 제1 피관리 서버 인스턴스가 상기 특정 세션을 로드할 수 없는 경우, 상기 특정 세션을 로드할 수 있는 상기 복수의 피관리 서버 인스턴스들 중 다른 피관리 서버 인스턴스의 표시를 제공하고, 상기 요청을 프로세싱하기 위해, 상기 특정 세션이 상기 복수의 피관리 서버 인스턴스들 중 제2 피관리 서버 인스턴스에서 로드되도록 지시하는 동작을 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법.

청구항 8

제7항에 있어서,

패치 프로세스의 모니터링 및 에러 처리를 제공하는 오케스트레이션 프레임워크를 이용하는 단계를 더 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법.

청구항 9

제7항에 있어서,

상기 어플리케이션 서버 환경은 Java EE 어플리케이션 서버를 포함하고, 각각의 리소스 그룹 템플릿은, 하나 이상의 관련 어플리케이션들 및 상기 하나 이상의 관련어플리케이션들이 의존하는 리소스들을 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법.

청구항 10

제7항에 있어서,

상기 멀티테넌트 어플리케이션 서버 환경 내에서 패치를 지원하기 위해 노드 관리자를 이용하는 단계를 더 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법.

청구항 11

제7항에 있어서,

상기 방법은 클러스터화된 환경에서 노드들에 패치들을 적용하기 위해 이용되는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법.

청구항 12

제7항에 있어서,

상기 방법은, 클라우드 환경 내에서 동작하는 복수의 테넌트들을 지원하기 위해 상기 클라우드 환경 내에서 수행되는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 방법.

청구항 13

명령어들이 저장된 비일시적 컴퓨터 판독가능 저장 매체로서, 상기 명령어들은 하나 이상의 컴퓨터들에 의해 판독 및 실행될 때, 상기 하나 이상의 컴퓨터들로 하여금 단계들을 수행하게 하며, 상기 단계들은:

복수의 파티션들을 지원하는 도메인의 일부로서 어플리케이션 서버 환경 내에서 작동하는 복수의 피관리(managed) 서버 인스턴스들과, 상기 어플리케이션 서버 환경 내에서 이용될 수 있는 복수의 전개가능한 리소스들과, 도메인 내에서 전개가능한 리소스들의 그룹들을 정의하는 하나 이상의 리소스 그룹 템플릿들과, 그리고 하나 이상의 파티션들을 제공하는 단계 - 각각의 파티션은 상기 도메인의 어드미니스트레이티브 및 런타임 서브디비전을 제공함 - 와; 그리고

테넌트(tenant)에 의한 이용을 위해, 하나 이상의 파티션들 및 리소스 그룹들을 상기 테넌트와 관련시키는 단계를 포함하고,

시스템은 인터럽션이 없이 상기 도메인의 오퍼레이션을 유지하는 제어된 방식으로 상기 복수의 피관리 서버 인스턴스들을 업데이트하도록 패치들을 적용하기 위해 어플리케이션 서버 환경에 구축된 가용성 특징들(availability features)의 장점을 취하는 패치 특징을 포함하며, 상기 시스템은,

상기 복수의 피관리 서버 인스턴스들에 대해 패치된 디렉토리로서 패치의 롤아웃(rollout)을 수행하는 동작,

각각의 피관리 서버 인스턴스에 대해, 피관리 서버 인스턴스를 셧-다운하고, 상기 패치된 디렉토리의 인스턴스를 사용하도록 상기 피관리 서버 인스턴스를 업데이트하고, 상기 피관리 서버 인스턴스를 업데이트하도록 상기 피관리 서버 인스턴스를 다시 시작하는 동작;

상기 복수의 피관리 서버 인스턴스들에 대한 패치의 롤아웃 동안, 특정 세션과 관련된 요청을 수신하면, 상기 요청을 프로세싱하기 위해 상기 복수의 피관리 서버 인스턴스들 중 제1 피관리 서버 인스턴스에서 상기 특정 세션을 로드하려고 시도하는 동작; 그리고

상기 복수의 피관리 서버 인스턴스들 중 제1 피관리 서버 인스턴스가 상기 특정 세션을 로드할 수 없는 경우, 상기 특정 세션을 로드할 수 있는 상기 복수의 피관리 서버 인스턴스들 중 다른 피관리 서버 인스턴스의 표시를 제공하고, 상기 요청을 프로세싱하기 위해, 상기 특정 세션이 상기 복수의 피관리 서버 인스턴스들 중 제2 피관리 서버 인스턴스에서 로드되도록 지시하는 동작을 포함하는 것을 특징으로 하는 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 14

제13항에 있어서,

패치 프로세스의 모니터링 및 에러 처리를 제공하는 오케스트레이션 프레임워크를 이용하는 단계를 더 포함하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 15

제13항에 있어서,

상기 어플리케이션 서버 환경은 Java EE 어플리케이션 서버를 포함하고, 각각의 리소스 그룹 템플릿은, 하나 이상의 관련 어플리케이션들 및 상기 하나 이상의 관련어플리케이션들이 의존하는 리소스들을 포함하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 16

제13항에 있어서,

상기 멀티테넌트 어플리케이션 서버 환경 내에서 패치를 지원하기 위해 노드 관리자를 이용하는 단계를 더 포함하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 17

제13항에 있어서,

상기 단계들은 클러스터화된 환경에서 노드들에 패치들을 적용하기 위해 이용되는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 18

제13항에 있어서,

상기 단계들은 클라우드 환경 내에서 동작하는 복수의 테넌트들을 지원하기 위해 상기 클라우드 환경 내에서 수행되는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 19

하나 이상의 컴퓨터 시스템들 상에서 실행되기 위한 명령어들을 포함하는 컴퓨터 판독가능한 저장 매체에 포함된 컴퓨터 프로그램으로서, 상기 명령어들은 실행될 때, 상기 하나 이상의 컴퓨터 시스템들이 청구항 제7항 내지 제12항 중 어느 한 항의 방법을 수행하도록 하는 것을 특징으로 하는 컴퓨터 프로그램.

청구항 20

삭제

발명의 설명

기 술 분 야

[0001] 저작권 공지

[0002] 본 명세서에서 개시된 부분은 저작권 보호를 받는 내용을 포함한다. 저작권자는 미국특허상표청의 특허 파일 또는 기록에 나타난 대로 본 특허 문서 또는 특허 개시내용을 어느 누군가가 팩시밀리 재생하는 것은 반대하지 않지만, 그 밖의 모든 것은 저작권으로 보호된다.

[0003] 우선권 주장 및 관련 출원의 상호 참조

[0004] 본 출원은 2014년 9월 24일자로 출원된 발명의 명칭이 "SYSTEM AND METHOD FOR MULTITENANT-AWARE PATCHING IN A MULTITENANT APPLICATION SERVER ENVIRONMENT"인 미국 가출원 제62/054,903호의 우선권의 이득을 주장하고, 2015년 1월 21일자로 출원된 발명의 명칭이 "SYSTEM AND METHOD FOR SUPPORTING MULTI-TENANCY IN AN APPLICATION SERVER, CLOUD, OR OTHER ENVIRONMENT"인 미국 특허 출원 제14/601,883호에 관계되며, 상기 출원

들 각각은 본 명세서에 참조로서 포함된다.

[0005] 기술분야

[0006] 본 발명의 실시예들은 일반적으로, 어플리케이션 서버들 및 클라우드 플랫폼 환경들에 관한 것이며, 특히 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템 및 방법에 관한 것이다.

배 경 기 술

[0007] 어플리케이션 서버 및 다른 엔터프라이즈 컴퓨팅 환경들에서, 어드미니스트레이터(administrator)에 대한 일반적인 태스크는 복수의 도메인들을 지원하는 일련의 어플리케이션 서버 인스톨레이션들을 패치하는 것을 필요로 한다. 패치는 특정한 문제들에 대한 일회성 수정이나 주기적인 버전 업데이트를 포함할 수 있다. 어째서 패치가 인스톨될 필요가 있는지와는 관계없이, 어드미니스트레이터는 일반적으로 어플리케이션 다운타임(downtime)을 최소화하면서 패치를 롤아웃(rollout)하기 위해 도메인의 각 노드에서 복잡한 일련의 단계들을 수행해야 하는 바, 이 단계들은 패치 환경이 각 호스트 상에서 최신 상태가 되도록 하는 단계와, 호스트 상에서 실행되는 그러한 서버들을 셧다운(shut down)하는 단계와, 그 다음, 어플리케이션 서버 인스턴스를 패치하고 재시작하며, 패치가 정확하게 작동하는지 검증하는 단계를 포함한다. 패치는 복잡한 프로세스고 심지어는 단일 어플리케이션 서버 인스턴스의 경우에도 수 분(many minutes)(도메인의 모든 노드에 패치가 적용될 때에는 수 시간이 될 수 있음)을 소요할 수 있기 때문에, 프로세스가 시스템 다운타임 가능성의 위험을 갖는 사용자에게 불안감을 줄 수 있다.

발명의 내용

[0008] 하나의 실시예에 따르면, 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템 및 방법이 본 명세서에 기술된다. 상기 시스템은 테넌트에 의한 사용을 위해 상기 테넌트와 하나 이상의 파티션(partition)들을 관련시킬 수 있고, 파티션은 도메인의 런타임 및 어드미니스트레이티브 서브디비전(runtime and administrative subdivision) 또는 슬라이스(slice)이다. 패치 프로세스는 어플리케이션 서버 클러스터링 환경에 의해 제공되는 고 가용성 특징들의 장점을 취하여, 인터럽션(interruption)이 없이 또는 다운타임이 영(zoro)인 채로 동작할 수 있게 하는 도메인의 능력을 유지하는 제어된 롤링 재시작에서 패치를 적용할 수 있다. 이 프로세스는 패치되지 않은 또는 이전 버전의 어플리케이션 서버, 어플리케이션 또는 가능한 롤백을 위한 다른 소프트웨어 컴포넌트를 보존하는 것 또는 복구불가능한 에러의 이벤트 시 자동 리버전을 제공하는 것을 포함하는 복잡한 또는 장기간 실행되는 태스크들을 자동화하기 위해 이용될 수 있다.

도면의 간단한 설명

[0009] 도 1은 하나의 실시예에 따른 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시(multi-tenancy)를 지원하는 시스템을 도시한다.

도 2는 하나의 실시예에 따른 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 더 도시한다.

도 3은 하나의 실시예에 따른 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 더 도시한다.

도 4는 하나의 실시예에 따른 예시적인 멀티테넌트 환경에서 이용하기 위한 도메인 구성(domain configuration)을 도시한다.

도 5는 하나의 실시예에 따른 예시적인 멀티-테넌트 환경을 더 도시한다.

도 6은 하나의 실시예에 따른 패치에 대한 지원을 도시한다.

도 7은 하나의 실시예에 따른 세션 처리(session handling)에 대한 지원을 포함하는 패치를 위한 시스템을 더 도시한다.

도 8은 하나의 실시예에 따른 세션 호환 검출에 대한 지원을 포함하는 패치를 위한 시스템을 더 도시한다.

도 9는 하나의 실시예에 따른 패치를 위한 시스템을 더 도시한다.

도 10은 하나의 실시예에 따른 패치를 위한 시스템을 더 도시한다.

도 11은 하나의 실시예에 따른 패치를 위한 시스템을 더 도시한다.

도 12는 하나의 실시예에 따른 패치를 위한 시스템을 더 도시한다.

도 13은 하나의 실시예에 따른 패치 이벤트 다이어그램을 도시한다.

도 14는 하나의 실시예에 따른 다른 패치 이벤트 다이어그램을 도시한다.

도 15는 하나의 실시예에 따른 다른 패치 이벤트 다이어그램을 도시한다.

도 16은 하나의 실시예에 따른 패치를 위한 방법의 순서도를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0010] 하나의 실시예에 따르면, 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템 및 방법이 본 명세서에 기술된다. 상기 시스템은 테넌트에 의한 사용을 위해 상기 테넌트와 하나 이상의 파티션들을 관련시킬 수 있고, 파티션은 도메인의 런타임 및 어드미니스트레이티브 서브디비전 또는 슬라이스이다. 패치 프로세스는 어플리케이션 서버 클러스터링 환경에 의해 제공되는 고 가용성 특징들의 장점을 취하여, 인터럽션이 없이 또는 다운타임이 영인 채로 동작할 수 있게 하는 도메인의 능력을 유지하는 제어된 롤링 재시작에서 패치를 적용할 수 있다. 이 프로세스는 패치되지 않은 또는 이전 버전의 어플리케이션 서버, 어플리케이션 또는 가능한 롤백을 위한 다른 소프트웨어 컴포넌트를 보존하는 것 또는 복구불가능한 에러의 이벤트 시 자동 리버전을 제공하는 것을 포함하는 복잡한 또는 장기간 실행되는 태스크들을 자동화하기 위해 이용될 수 있다.

[0011] 어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경

[0012] 도 1은 하나의 실시예에 따른 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 도시한다.

[0013] 도 1에 도시된 바와 같이, 하나의 실시예에 따르면, 어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경(100) 또는, 소프트웨어 어플리케이션들의 개발 및 실행을 할 수 있게 하는 다른 컴퓨팅 환경은 어플리케이션 서버 도메인을 정의하기 위해 런타임 시 이용되는 도메인(102) 구성을 포함하고 이에 따라 동작하도록 구성될 수 있다.

[0014] 하나의 실시예에 따르면, 어플리케이션 서버는 런타임 시 이용하기 위해 정의되는 하나 이상의 파티션들(104)을 포함할 수 있다. 각각의 파티션은 글로벌하게 고유한 파티션 식별자(ID) 및 파티션 구성과 관련될 수 있고, 또한 리소스 그룹 템플릿(126)에 대한 참조와 함께 하나 이상의 리소스 그룹들(124) 및/또는 파티션별(partition-specific) 어플리케이션들 및 리소스들(128)을 포함할 수 있다. 도메인-레벨 리소스 그룹들, 어플리케이션들 및/또는 리소스들(140)은 또한, 옵션에 따라서는 리소스 그룹 템플릿을 참조하여 도메인 레벨에서 정의될 수 있다.

[0015] 각각의 리소스 그룹 템플릿(160)은 하나 이상의 어플리케이션들 A(162), B(164), 리소스들 A(166), B(168) 및/또는 다른 전개가능한 어플리케이션들 또는 리소스들(170)을 정의할 수 있고, 리소스 그룹에 의해 참조될 수 있다. 예를 들어, 도 1에 도시된 바와 같이, 파티션(104) 내의 리소스 그룹(124)은 리소스 그룹 템플릿(160)을 참조(190)할 수 있다.

[0016] 일반적으로, 시스템 어드미니스트레이터는 파티션들, 도메인-레벨 리소스 그룹들 및 리소스 그룹 템플릿들 및 보안 영역(security realms)을 정의할 수 있고, 파티션 어드미니스트레이터는 예컨대, 파티션-레벨 리소스 그룹들을 생성, 파티션에 어플리케이션들을 전개(deploy) 또는 파티션에 대한 특정 영역을 참조함으로써, 자신만의 파티션의 양상들을 정의할 수 있다.

[0017] 도 2는 하나의 실시예에 따른 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 더 도시한다.

[0018] 도 2에 예시된 바와 같이, 하나의 실시예에 따르면, 파티션(202)은 예컨대, 리소스 그룹 템플릿(210)에 대한 참조(206)를 포함하는 리소스 그룹(205), 가상 타겟(예컨대, 가상 호스트) 정보(204) 및 플러그가능 데이터베이스(PDB: pluggable database) 정보(208)를 포함할 수 있다. 리소스 그룹 템플릿(예컨대, 210)은 예컨대, 자바 어드미니스트레이터 서버(JMS) 서버(213), 축적 전송(SAF: store-and-forward) 에이전트(215), 메일 세션 컴포넌트(216) 또는 자바 데이터베이스 연결(JDBC) 리소스(217)과 같은 리소스들과 함께 복수의 어플리케이션들 A(211) 및 B(212)을 정의할 수 있다.

[0019] 도 2에 도시된 리소스 그룹 템플릿이 예로서 제공되며, 다른 실시예들에 따르면 다른 타입의 리소스 그룹 템플

릿들 및 요소들이 제공될 수 있다.

[0020] 하나의 실시예에 따르면, 파티션(예컨대, 202) 내의 리소스 그룹이 특정한 리소스 그룹 템플릿(예컨대, 210)을 참조(220)할 때, 특정한 파티션과 관련된 정보는 파티션별 정보(230), 예컨대 파티션별 PDB 정보를 나타내기 위해, 참조된 리소스 그룹 템플릿과 조합하여 이용될 수 있다. 그 다음, 파티션별 정보는 파티션에 의해 사용하기 위한 리소스들, 예컨대 PDB 리소스를 구성하기 위해 어플리케이션 서버에 의해 이용될 수 있다. 예를 들어, 파티션(202)과 관련된 파티션별 PDB 정보는 그 파티션에 의해 이용하기 위한 적절한 PDB(238)를 갖는 컨테이너 데이터베이스(CDB)(236)를 구성(232)하기 위해 어플리케이션 서버에 의해 이용될 수 있다.

[0021] 마찬가지로, 하나의 실시예에 따르면, 특정한 파티션과 관련된 가상 타겟 정보는 파티션에 의해 이용하기 위한 파티션별 가상 타겟(240)(예컨대, baylandurgentcare.com)을 정의(239)하기 위해 이용될 수 있는 바, 이는 그 다음, URL(uniform resource locator), 예컨대, http://baylandurgentcare.com을 통해 액세스가능하게 될 수 있다.

[0022] 도 3은 하나의 실시예에 따른 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 더 도시한다.

[0023] 하나의 실시예에 따르면, config.xml 구성 파일과 같은 시스템 구성이 파티션을 정의하기 위해 이용되는 바, 이 시스템 구성은 그 파티션 및/또는 다른 파티션 속성들과 관련된 리소스 그룹들에 대한 구성 요소들을 포함한다. 속성 이름/값 쌍들을 이용하여 파티션 당 값들이 특정될 수 있다.

[0024] 하나의 실시예에 따르면, 복수의 파티션들은 피관리 서버/클러스터(242) 내에서 또는 CDB(243)에게로의 액세스를 제공할 수 있고 웹 티어(web tier)(244)를 통해 액세스가능한 유사한 환경 내에서 실행될 수 있다. 이는 예컨대, 도메인 또는 파티션이 (CDB의) PDB들 중 하나 이상과 관련되도록 한다.

[0025] 하나의 실시예에 따르면, 이 예시적인 파티션 A(250) 및 파티션 B(260)에서 복수의 파티션들 각각은 그 파티션과 관련된 복수의 리소스들을 포함하도록 구성될 수 있다. 예를 들어, 파티션 A는 PDB A(259)와 관련된 데이터소스 A(257)와 함께 어플리케이션 A1(252), 어플리케이션 A2(254) 및 JMS A(256)를 포함하고 리소스 그룹(251)을 포함하도록 구성될 수 있고, 파티션은 가상 타겟 A(258)를 통해 액세스가능하다. 마찬가지로, 파티션 B(260)는 PDB B(269)와 관련된 데이터소스 B(266)와 함께 어플리케이션 B1(262), 어플리케이션 B2(264) 및 JMS B(266)를 포함하는 리소스 그룹(261)을 포함하도록 구성될 수 있고, 파티션은 가상 타겟 B(268)을 통해 액세스가능하다.

[0026] 상기 여러 예들은 CDB 및 PDB들의 이용을 예시하지만, 다른 실시예들에 따르면 다른 타입의 멀티테넌트 또는 비-멀티테넌트(non-multi-tenant) 데이터베이스들이 지원될 수 있고, 예컨대 스키마들(schemas)의 이용 또는 서로 다른 데이터베이스들의 이용을 통해 각각의 파티션에 대해 특정한 구성이 제공될 수 있다.

[0027] 리소스들

[0028] 하나의 실시예에 따르면, 리소스는 환경의 도메인에 대해 전개될 수 있는 시스템 리소스, 어플리케이션 또는 다른 리소스 또는 객체이다. 예를 들어, 하나의 실시예에 따르면, 리소스는 서버, 클러스터 또는 다른 어플리케이션 서버 타겟에 대해 전개될 수 있는 어플리케이션, JMS, JDBC, 자바메일, WLDF, 데이터 소스 또는 다른 시스템 리소스 또는 다른 타입의 객체일 수 있다.

[0029] 파티션들

[0030] 하나의 실시예에 따르면, 파티션은, 파티션 식별자(ID) 및 구성과 관련될 수 있고 어플리케이션들을 포함하고 그리고/또는 리소스 그룹들 및 리소스 그룹 템플릿들의 이용을 통해 도메인 범위의(domain-wide) 리소스들을 참조할 수 있는 도메인의 런타임 및 어드미니스트레이티브 서브디비전 또는 슬라이드이다.

[0031] 일반적으로, 파티션은 자신만의 어플리케이션들을 포함하고, 리소스 그룹 템플릿들을 통해 도메인 범위의 어플리케이션들을 참조하며, 자신만의 구성을 가질 수 있다. 파티션가능한 엔티티들은 리소스들, 예컨대 JMS, JDBC, 자바메일, WLDF 리소스들, 및 JNDI 이름공간, 네트워크 트래픽, 작업 관리자들 및 보안 정책 및 영역들과 같은 다른 컴포넌트들을 포함할 수 있다. 멀티테넌트 환경의 맥락에서, 시스템은 테넌트와 관련된 파티션들의 어드미니스트레이티브 및 런타임 양상들(administrative and runtime aspects)에게 테넌트 액세스를 제공하도록 구성될 수 있다.

[0032] 하나의 실시예에 따르면, 파티션 내의 각각의 리소스 그룹은 옵션에 따라서는, 리소스 그룹 템플릿을 참조할 수

있다. 파티션은 복수의 리소스 그룹들을 가질 수 있고, 이들 각각은 리소스 그룹 템플릿을 참조할 수 있다. 각각의 파티션은 파티션의 리소스 그룹들이 참조하는 리소스 그룹 템플릿들에 특정되지 않는 구성 데이터에 대한 속성들을 정의할 수 있다. 이는 파티션이 그 파티션으로 이용할 특정값들에 대한 리소스 그룹 템플릿에 정의된 전개가능한 리소스들의 바인딩으로서 역할을 하게 한다. 일부 경우들에서, 파티션은 리소스 그룹 템플릿에 의해 특정되는 구성 정보를 오버라이드(override)할 수 있다.

[0033] 하나의 실시예에 따르면, 예컨대 config.xml 구성 파일에 의해 정의된 파티션 구성은 복수의 구성 요소들, 예컨대, 파티션을 정의하는 속성들 및 차일드 요소들을 포함하는 "partition", 상기 파티션에 대해 전개되는 어플리케이션들 및 리소스들을 포함하는 "resource-group", 해당 템플릿에 의해 정의되는 어플리케이션들 및 리소스들을 포함하는 "resource-group-template", 데이터베이스별 서비스 이름, 사용자 이름 및 패스워드를 포함하는 "jdbc-system-resource-override" 및 리소스 그룹 템플릿들 내의 매크로 교체(macro replacement)를 위해 이용될 수 있는 속성 키 값들을 포함하는 "partition-properties"을 포함할 수 있다.

[0034] 시동 시, 시스템은 리소스 그룹 템플릿으로부터 각각의 리소스에 대한 파티션별 구성 요소들을 생성하기 위해 구성 파일에 의해 제공되는 정보를 이용할 수 있다.

[0035] 리소스 그룹들

[0036] 하나의 실시예에 따르면, 리소스 그룹은 도메인 또는 파티션 레벨에서 정의될 수 있고 리소스 그룹 템플릿을 참조할 수 있는 전개가능한 리소스들의 명명된, 완전히 적격한 집합(named, fully-qualified collection)이다. 리소스 그룹 내의 리소스들은 어드미니스트레이터가 시작 또는 이 리소스들에 연결하기 위해 필요한 모든 정보, 예컨대 데이터 소스에 연결하기 위한 크리덴셜들 또는 어플리케이션에 대한 타겟팅 정보를 제공했다는 점에서 완전히 적격한(fully-qualified) 것으로 고려된다.

[0037] 시스템 어드미니스트레이터는 도메인 레벨에서 또는 파티션 레벨에서 리소스 그룹들을 선언할 수 있다. 도메인 레벨에서, 리소스 그룹은 그룹 관련 리소스들에 편리한 방식을 제공한다. 시스템은 비그룹화된 리소스들과 동일하게 도메인-레벨 리소스 그룹에서 선언된 리소스들을 관리할 수 있어서, 상기 리소스들은 시스템 시동 동안 시작되고, 시스템 셧-다운 동안 정지될 수 있다. 어드미니스트레이터는 또한, 그룹 내의 리소스를 개별적으로 정지, 시작 또는 제거할 수 있고, 그룹 상에서 동작함으로써 묵시적으로 상기 그룹 내의 모든 리소스들에 영향을 줄 수 있다. 예를 들어, 리소스 그룹을 정지시키는 것은 아직 정지되지 않은 그룹 내의 리소스들 모두를 정지시키고, 리소스 그룹을 시작시키는 것은 아직 시작되지 않은 그룹 내의 어떤 리소스들을 시작시키며, 리소스 그룹을 제거하는 것은 그룹에 포함된 리소스들 모두를 제거한다.

[0038] 파티션 레벨에서, 시스템 또는 파티션 어드미니스트레이터는 어떤 보안 제약들을 겪는 파티션 내의 0개 이상의 리소스 그룹들을 구성할 수 있다. 예를 들어, SaaS 사용 경우에서, 다양한 파티션-레벨 리소스 그룹들은 도메인-레벨 리소스 그룹을 참조할 수 있고, PaaS 사용 경우에서, 리소스 그룹 템플릿을 참조하는 것이 아니라, 해당 파티션 내에서만 이용가능해지는 어플리케이션들 및 이들의 관련 리소스들을 표시하는 파티션-레벨 리소스 그룹들이 생성될 수 있다.

[0039] 하나의 실시예에 따르면, 리소스 그룹화는 어플리케이션들 및 이 어플리케이션들이 도메인 내에서 별개의 어드미니스트레이티브 유닛으로서 이용하는 리소스들을 함께 그룹화하기 위해 이용될 수 있다. 예를 들어, 하기 기술되는 의료 기록들(MedRec) 어플리케이션에서, 리소스 그룹화는 MedRec 어플리케이션 및 이의 리소스들을 정의한다. 복수의 파티션들은 각각 파티션별 구성 정보를 이용하여 동일한 MedRec 리소스 그룹을 실행할 수 있어서, 각각의 MedRec 인스턴스의 일부인 어플리케이션들은 각각의 파티션에 특정적이게 된다.

[0040] 리소스 그룹 템플릿들

[0041] 하나의 실시예에 따르면, 리소스 그룹 템플릿은 리소스 그룹으로부터 참조될 수 있는 도메인 레벨에서 정의되는 전개가능한 리소스들의 집합이며, 이의 리소스들을 활성화시키기 위해 요구되는 정보의 일부는 파티션 레벨 구성의 사양을 지원할 수 있도록 템플릿 자체의 일부로서 저장되지 않을 수 있다. 도메인은 어떤 수의 리소스 그룹 템플릿들을 포함할 수 있고, 이 템플릿들 각각은 예컨대, 하나 이상의 관련 자바 어플리케이션들 및 이 어플리케이션들이 의존하는 리소스들을 포함할 수 있다. 이러한 리소스들에 관한 정보의 일부는 모든 파티션들에 걸쳐 동일할 수 있고, 다른 정보는 파티션 마다 다양할 수 있다. 모든 구성이 도메인 레벨에서 특정될 필요는 없는바 - 대신 파티션 레벨 구성이 매크로들 또는 속성 이름/값 쌍들의 이용을 통해 리소스 그룹 템플릿에 특정될 수 있다.

[0042] 하나의 실시예에 따르면, 특정한 리소스 그룹 템플릿은 하나 이상의 리소스 그룹들에 의해 참조될 수 있다. 일

반적으로, 어떤 소정 파티션 내에서, 리소스 그룹 템플릿은 오직 한 번에 하나의 리소스 그룹에 의해 참조될 수 있는 바, 즉 동일한 파티션 내에서 복수의 리소스 그룹들에 의해 동시에 참조될 수 없다. 그러나, 리소스 그룹 템플릿은 다른 파티션 내의 다른 리소스 그룹에 의해 동시에 참조될 수 있다. 리소스 그룹을 포함하는 객체, 예컨대 도메인 또는 파티션은 리소스 그룹 템플릿 내의 어떤 토큰들의 값을 설정하기 위해 속성 이름/값 할당들을 이용할 수 있다. 시스템이 참조 리소스 그룹을 이용하여 리소스 그룹 템플릿을 활성화시킬 때, 이는 이 토큰들을 리소스 그룹 포함 객체에 설정된 값들과 교체할 수 있다. 일부 경우들에서, 시스템은 또한, 각각의 파티션/템플릿 조합에 대해 런타임 구성을 생성하기 위해 통계적으로 구성된 리소스 그룹 템플릿들 및 파티션들을 이용할 수 있다.

[0043] 예를 들어, SaaS 사용 경우에서, 시스템은 동일한 어플리케이션들 및 리소스들을 복수번 활성화시킬 수 있는 바, 이는 이 어플리케이션들 및 리소스들을 이용할 각각의 파티션에 대해 한번씩 활성화시키는 것을 포함한다. 어드미니스트레이터가 리소스 그룹 템플릿을 정의할 때, 이들은 다른 곳에 공급될 정보를 표시하기 위해 토큰들을 이용할 수 있다. 예를 들어, CRM-관련 데이터 리소스에 연결할 시 이용하기 위한 사용자이름이 \${CRMDataUsername}로서 리소스 그룹 템플릿에 표시될 수 있다.

[0044] 테넌트들

[0045] 하나의 실시예에 따르면, 멀티-테넌트(MT) 어플리케이션 서버 환경과 같은 멀티-테넌트 환경에서, 테넌트는 하나 이상의 파티션들 및/또는 하나 이상의 테넌트-인지 어플리케이션들에 의해 표시될 수 있거나 또는 그렇지 않으면 이와 관련될 수 있는 엔티티이다.

[0046] 예를 들어, 테넌트들은 서로 다른 외부의 회사들 또는 특정 기업 내의 서로 다른 부서들(예컨대, HR 및 재무 부서들)과 같은 별개의 사용자 조직들을 나타낼 수 있고, 이들 각각은 서로 다른 파티션과 관련될 수 있다. 글로벌 하게 고유한 테넌트 신원(테넌트 ID)는 특정한 시점에 특정한 테넌트와의 특정한 사용자의 관련성이다. 시스템은 예컨대, 사용자 신원 저장소를 참조함으로써 특정한 사용자가 어느 테넌트에 속하는지를 사용자 신원으로부터 도출할 수 있다. 사용자 신원은 시스템으로 하여금 이들로만 한정되는 것은 아니지만, 사용자가 속할 수 있는 테넌트를 포함하여 사용자가 수행하도록 인가된 그러한 동작(action)들을 실시할 수 있게 한다.

[0047] 하나의 실시예에 따르면, 시스템은 서로 다른 테넌트들의 관리 및 런타임이 서로 격리되게 한다. 예를 들어, 테넌트들은 자신의 어플리케이션들 및 이들이 액세스할 수 있는 리소스들의 일부 거동들을 구성할 수 있다. 시스템은 특정한 테넌트가 다른 테넌트에 속하는 아티팩트들을 어드미니스터(administer)할 수 없게 할 수 있고, 런타임 시 특정한 테넌트 대신 작동하는 어플리케이션들이 그 테넌트와 관련된 리소스들만 참조하고 다른 테넌트들과 관련된 리소스들을 참조하지 못하게 할 수 있다.

[0048] 하나의 실시예에 따르면, 테넌트-비인지 어플리케이션은 테넌트들을 분명하게(explicitly) 다루는 로직을 포함하지 않는 어플리케이션이어서, 상기 어플리케이션이 이용하는 어떤 리소스들은 어떤 사용자가 상기 어플리케이션이 응답하는 요청을 제출했는지에 관계없이 액세스가능할 수 있다. 이와는 대조적으로, 테넌트-인지 어플리케이션은 테넌트들을 분명하게 다루는 로직을 포함한다. 예를 들어, 사용자의 신원에 기초하여, 어플리케이션은 사용자가 속하는 테넌트를 도출할 수 있고, 그 정보를 테넌트별 리소스들에 액세스하기 위해 이용할 수 있다.

[0049] 하나의 실시예에 따르면, 시스템은 사용자들이 테넌트-인지형이도록 명시적으로 작성되는 어플리케이션들을 전개할 수 있게 하여서, 어플리케이션 개발자들은 현재의 테넌트의 테넌트 ID를 획득할 수 있다. 그 다음, 테넌트-인지 어플리케이션은 상기 어플리케이션의 단일 인스턴스를 이용하는 복수의 테넌트들을 처리하기 위해 테넌트 ID를 이용할 수 있다.

[0050] 예를 들어, 단일 의사의 사무실 또는 병원을 지원하는 MedRec 어플리케이션은 두 개의 서로 다른 파티션들 또는 테넌트들, 예컨대 Bayland Urgent Care 테넌트 및 Valley Health 테넌트에 노출될 수 있는 바, 이들 각각은 기저 어플리케이션 코드를 변경함이 없이 예컨대, 별개의 PDB들과 같은 별개의 테넌트별 리소스들에 액세스할 수 있다.

[0051] 예시적인 도메인 구성 및 멀티-테넌트 환경

[0052] 하나의 실시예에 따르면, 어플리케이션들은 도메인 레벨에서 리소스 그룹 템플릿에 대해 또는 파티션 범주의(scoped to) 또는 도메인 범위의 리소스 그룹에 대해 전개될 수 있다. 어플리케이션 구성은 어플리케이션 당 또는 파티션 당 특정되는 전개 계획들을 이용하여 오버라이드될 수 있다. 전개 계획들은 또한, 리소스 그룹의 일부로서 특정될 수 있다.

[0053] 도 4는 하나의 실시예에 따른 예시적인 멀티-테넌트 환경에서 이용하기 위한 도메인 구성을 도시한다.

[0054] 하나의 실시예에 따르면, 시스템이 파티션을 시작할 때, 이는 제공되는 구성에 따라 각각의 데이터베이스 인스턴스들에 대해 가상 타겟들(예컨대, 가상 호스트들) 및 연결 풀들을 생성하는 바, 이러한 생성은 각각의 파티션당 하나를 생성하는 것을 포함한다.

[0055] 전형적으로는, 각각의 리소스 그룹 템플릿은 하나 이상의 관련 어플리케이션들 및 이 어플리케이션들이 의존하는 리소스들을 포함할 수 있다. 각각의 파티션은 파티션과 관련된 특정 값들에 대한 리소스 그룹 템플릿들 내의 전개가능한 리소스들의 바인딩을 제공함으로써 상기 파티션이 참조하는 리소스 그룹 템플릿에 특정되지 않은 구성 데이터를 제공할 수 있는 바, 이는 일부 경우들에서, 상기 리소스 그룹 템플릿에 의해 특정된 특정 구성 정보를 오버라이드하는 것을 포함한다. 이는 시스템이 각각의 파티션이 정의한 속성 값들을 이용하여 각각의 파티션에 대해 서로 다르게 리소스 그룹 템플릿에 의해 표시되는 어플리케이션을 활성화시킬 수 있게 한다.

[0056] 일부 예들에서, 파티션은, 리소스 그룹 템플릿들을 참조하지 않거나 또는 자신만의 파티션 범주의 전개가능한 리소스들을 직접적으로 정의하는 리소스 그룹들을 포함할 수 있다. 파티션 내에 정의되는 어플리케이션들 및 데이터 소스들은 일반적으로, 그 파티션에게만 이용가능하다. 리소스들은 파티션들, <partitionName>/<resource JNDI name>, 또는 domain:<resource JNDI name>를 이용하여 파티션들에 걸쳐 액세스될 수 있도록 전개될 수 있다.

[0057] 예를 들어, MedRec 어플리케이션은 복수의 자바 어플리케이션들, 데이터 소스, JMS 서버 및 메일 세션을 포함할 수 있다. 복수의 테넌트들에 대해 MedRec 어플리케이션을 실행하기 위해, 시스템 어드미니스트레이터는 단일 MedRec 리소스 그룹 템플릿(286)을 정의할 수 있고, 상기 템플릿에 그러한 전개가능한 리소스들을 선언할 수 있다.

[0058] 도메인 레벨의 전개가능한 리소스들과는 대조적으로, 리소스 그룹 템플릿에 선언된 전개가능한 리소스들은 템플릿에 완전히 구성되지 않을 수 있거나 또는 있는 그대로(as-is) 활성화되지 못할 수 있는 바, 그 이유는 이 리소스들에 일부 구성 정보가 결여되어 있기 때문이다.

[0059] 예를 들어, MedRec 리소스 그룹 템플릿은 어플리케이션들에 의해 이용되는 데이터 소스를 선언할 수 있지만, 이는 데이터베이스에 연결하기 위한 URL을 특정하지 않을 수 있다. 서로 다른 테넌트들과 관련된 파티션들, 예컨대 파티션 BUC-A(290)(Bayland Urgent Care, BUC) 및 파티션 VH-A(292)(Valley Health, VH)는 MedRec 리소스 그룹 템플릿을 참조(296, 297)하는 MedRec 리소스 그룹(293, 294)을 각각 포함함으로써 하나 이상의 리소스 그룹 템플릿들을 참조할 수 있다. 그 다음, 상기 참조는 Bayland Urgent Care 테넌트가 사용하기 위한 BUC-A 파티션과 관련된 가상 호스트 baylandurgentcare.com(304) 및 Valley Health 테넌트가 사용하기 위한 VH-A 파티션과 관련된 가상 호스트 valleyhealth.com(308)를 포함하는, 각각의 테넌트에 대한 가상 타겟들/가상 호스트들을 생성(302, 306)하기 위해 이용될 수 있다.

[0060] 도 5는 하나의 실시예에 따른 예시적인 멀티-테넌트 환경을 더 도시한다. 도 5에 도시된 바와 같이 그리고 하나의 실시예에 따라 두 파티션들이 MedRec 리소스 그룹 템플릿을 참조하는 상기 예를 계속 들어, 서블릿 엔진(310)이 복수의 테넌트 환경들, 이 예에서는 Bayland Urgent Care Physician 테넌트 환경(320) 및 Valley Health Physician 테넌트 환경(330)을 지원하기 위해 이용될 수 있다.

[0061] 하나의 실시예에 따르면, 각각의 파티션(321, 331)은 테넌트 환경에 대해 유입 트래픽을 수락할 서로 다른 가상 타겟, 및 파티션 및 이의 리소스들(324, 334)(이 예에서는 bayland urgent care 데이터베이스 또는 valley health 데이터베이스를 각각 포함)에 연결하기 위한 서로 다른 URL(322, 332)를 정의할 수 있다. 상기 데이터베이스 인스턴스들은 호환가능한 스키마를 이용할 수 있는 바, 그 이유는 동일한 어플리케이션 코드가 두 데이터베이스들에 대해 실행될 것이기 때문이다. 시스템이 파티션들을 시작할 때, 이는 가상 타겟들, 및 각각의 데이터베이스 인스턴스들에게로의 연결 풀들을 생성할 수 있다.

[0062] * * * * * * * * * * *

[0063] 멀티테넌트-인지 패치(multitenant-Aware Patching)

[0064] 하나의 실시예에 따르면, 멀티테넌트 어플리케이션 서버 환경에서 패치를 지원하는 시스템 및 방법이 본 명세서에 기술된다. 상기 시스템은 테넌트에 의한 사용을 위해 상기 테넌트와 하나 이상의 파티션들을 관련시킬 수 있고, 파티션은 도메인의 런타임 및 어드미니스트레이티브 서브디비전 또는 슬라이스이다. 패치 프로세스는 어플리케이션 서버 클러스터링 환경에 의해 제공되는 고 가용성 특징들의 장점을 취하여, 인터럽션이 없이 또는 다

운타임이 영인 채로 동작할 수 있게 하는 도메인의 능력을 유지하는 제어된 롤링 재시작에서 패치를 적용할 수 있다. 이 프로세스는 패치되지 않은 또는 이전 버전의 어플리케이션 서버, 어플리케이션 또는 가능한 롤백을 위한 다른 소프트웨어 컴포넌트를 보존하는 것 또는 복구불가능한 에러의 이벤트 시 자동 리버전을 제공하는 것을 포함하는 복잡한 또는 장기간 실행되는 태스크들을 자동화하기 위해 이용될 수 있다.

[0065] 다양한 실시예에 따르면, 본 명세서에서 제공되는 패치 프로세스의 설명은 다음의 컨셉들 중 일부 또는 전부를 이용한다.

[0066] PSU(patch set update): 패치 세트 업데이트

[0067] ZDT(zero downtime): 제로 다운타임

[0068] 워크플로우: 오케스트레이션 프레임워크 또는 패치 오케스트레이터에 의해 실행되는 태스크들의 시퀀스.

[0069] 패치 프리미티브(patching primitive): 패치 롤아웃의 재사용가능한 부분을 나타내는 논리 연산(logical operation).

[0070] 아웃 오프 플레이스 패치(out of place patching): 예컨대, 프로덕션 서버들의 적은 다운타임을 요하고 필요한 경우 본래의 버전으로 더욱 쉽게 롤백할 수 있게 하는 능력을 제공하는 아웃 오브 밴드 패치 및 테스트(out of band patching and testing) 방식으로, 비프로덕션 서버 상에서 실행되고, 그 다음, 프로덕션 서버로 패치들을 밀어내기 전에 상기 패치들을 테스트 및 검증하는 Oracle Home의 패치.

[0071] 도 6은 하나의 실시예에 따른 패치에 대한 지원을 도시한다.

[0072] 도 6에 예시된 바와 같이, 하나의 실시예에 따르면, 시스템은, (MS1, MS2 및 MS3으로서 표시된) 피관리 서버들(managed servers)의 제1 페일오버(failover) 그룹(404) 및 (MS4, MS5 및 MS6로서 표시된) 피관리 서버들의 제2 페일오버 그룹을 포함하여, 피관리 서버 또는 클러스터를 어드미니스터링(adminstering)하는 역할을 하는 어드미니스트레이션 서버(어드민 서버)(400)를 포함할 수 있다. 어드미니스트레이션 서버는 REST API(410) 또는 다른 타입의 인터페이스를 통해 클라이언트들에 의해 액세스될 수 있다.

[0073] 하나의 실시예에 따르면, 시스템은 또한, 패치 오케스트레이션 프레임워크 또는 패치 오케스트레이터(420)을 포함하는 바, 이는 패치 워크플로우의 일부로서 하기에 더 기술되는 바와 같이 복수의 패치 프리미티브들을 이용하여 서로 다른 버전의 소프트웨어 컴포넌트들 또는 패치들을 롤아웃(roll out) 및/또는 적용하는 동작을 한다.

[0074] 일반적으로, 패치 오케스트레이터는 강건한 방식(robust manner)으로 동작하도록 그리고 태스크 재시도(retry) 및 롤백 시멘틱들과 같은 기능에 대한 지원을 포함하도록 설계된다.

[0075] 하나의 실시예에 따르면, 패치 오케스트레이션 프로세스는 어플리케이션 서버에 의해 제공되는 다양한 특징들을 레버리지(leverage)하여 고급 기능들(dvanced functionalities), 예컨대, 백워드 호환(backward-compatible)이 가능하지 않을 수 있는 어플리케이션 세션들을 처리하기 위한 능력, 서버를 셧 다운시키기 전에 종료시키기 위해 피관리 서버 내의 기존 세션들을 위해 대기하는 세션-인지 정상적 셧다운(session-aware graceful shutdown), 패치 윈도우 동안 복제된(replicated) 세션들의 자동 직렬화 해제를 턴 오프시키는 복제된 세션들의 지연 직렬화 해제(lazy de-serialization), 클러스터 재시작을 회피하기 위한 지연 직렬화 해제의 동적 턴 온/오프, 그룹 정보에 기초한 페일오버와 같은 기능들을 제공하는 바, 이들 각각의 특징 또는 기능은 하기에 더 설명된다.

[0076] 하나의 실시예에 따르면, 패치 오케스트레이터에 의해 지원되는 패치 프리미티브들의 예들은 트래픽 디렉터 또는 다른 타입의 로드 밸런서(430), 예를 들어 오라클 트래픽 디렉터(OTD)와 통신하여 특정 서버로의 트래픽을 중지시키는 서버 중지(quiesce server)(422), 새로운 타겟을 포인팅하기 위해 홈 디렉토리 또는 다른 스토리지(예컨대, 오라클 홈)의 심볼적 링크(symlink)를 변경시키는 홈 업데이트(424), 앱 레디(ready app) 또는 유사한 프레임워크와 통신하고, 모든 등록된 어플리케이션들이 레디 상태일 때만 완료되는 앱들 레디 체크(426), 및 특정 서버에 트래픽을 전송하는 것을 재개하기 위해 예컨대 OTD와 통신하는 활성화 서버(428)를 포함할 수 있다.

[0077] 하나의 실시예에 따르면, 패치 오케스트레이터는 자신의 프리미티브들 및 워크플로우와 함께, 예컨대 초기 패치된 또는 패치되지 않은(unpatched) 버전(452)으로부터 후속적으로 패치된 버전(454)으로 하나 이상의 피관리 서버들(451)에 대한 홈 디렉토리들 또는 다른 스토리지들(450)의 세트를 패치 또는 업데이트하도록 요구되는 정보를 포함하여 소프트웨어 컴포넌트들 또는 패치들의 서로 다른 버전들을 지원하기 위해 패치 데이터베이스(440)와 조합하여 이용될 수 있다.

[0078] 예를 들어, 도 6에 예시된 바와 같이, 클러스터는 상기에 기술된 바와 같이 피관리 서버들의 두 개의 페일오버 그룹들을 포함할 수 있고, 제1 페일오버 그룹 및 이의 선택된 피관리 서버들(MS1, MS2 및 MS3)의 선택은 패치된 버전의 홈 디렉토리를 이용하고, 제2 페일오버 그룹 및 다른 피관리 서버들(MS4, MS5 및 MS6)은 초기의 또는 패치되지 않은 버전의 홈 디렉토리를 이용한다.

[0079] 트래픽 디렉터 또는 로드 밸런서로부터의 요청은 페일오버 그룹 내의 어떤 서버에 페일오버할 수 있다. 하기에 더 기술되는 바와 같이, 하나의 실시예에 따르면, 지연 세션 직렬화 해제 기능은 두 개의 페일오버 그룹들 및 이들의 피관리 서버들에 걸친(span) 어떤 세션들의 페일오버를 정상적으로(gracefully) 처리하기 위해 이용될 수 있다.

[0080] 도 7은 하나의 실시예에 따른 세션 처리에 대한 지원을 포함하는 패치를 위한 시스템을 더 도시한다.

[0081] 전형적인 어플리케이션 서버 환경에서, 서버 인스턴스의 셧 다운 및 후속적인 재시작은 일부 시간을 소요할 수 있고, 아마 심지어는 몇 분을 소요할 수 있다. 이를 해소하기 위해, 하나의 실시예에 따르면, 시스템은, 활성 세션들이 상기 시스템 내의 다른 곳에 제공되는지를 결정하는 것과 그리고 만일 그렇지 않으면, 의도된 서버를 셧 다운시키기 전에 세션들을 이용가능하게 만드는 것을 포함하여, 셧 다운 시 수행될 수 있는 더욱 스마트한 세션 복제 프로세스를 포함한다.

[0082] 도 7에 예시된 바와 같이, 하나의 실시예에 따르면, 트래픽 디렉터는 로드 밸런싱(452), 503 해더 검출(454), 동적 디스커버리(456) 및 건강 체크(458)와 같은 기능을 지원하고, 어플리케이션 서버 클러스터링 환경(460)은 동적 지연 세션 직렬화 해제(462), 세션 패치(464) 및 고아 세션 정리(orphaned session cleanup)(468)과 같은 기능을 지원하고, 웹 컨테이너(470)는 세션 호환 검출(472)과 같은 기능을 지원하고, 그리고 서버 수명 컴포넌트(480)는 셧 다운시 세션 복제(480) 및 모든 세션 대기(484)와 같은 기능을 지원한다.

[0083] 하나의 실시예에 따르면, 상기 컴포넌트들 각각은, 패치 전과 후의 패치 지원의 동적 턴 온 및 오프, 세션 패치, 복수의 백업들을 회피하기 위한 고아 세션 클린업, 어떻게 하나의 서버가 트래픽 디렉터에 503 메시지를 전송하여 다른 서버를 시도하도록 할 수 있는지를 포함하는 비호환가능한 세션들의 처리, 및 복수의 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트의 처리와 같은 다양한 상황들을 해소하기 위한 이들의 사용을 포함하여 하기에 더욱 상세히 기술된다.

[0084] 예를 들어, 하나의 실시예에 따르면, 시스템은 서로 다른 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트들이 새로운 파티션을 생성하고 그리고 상기 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트의 다른 버전을 상기 새로운 파티션에서 셋업(set up)함으로써 다른 파티션들에 전개(deploy)될 수 있게 한다. 트래픽 디렉터는, 얼마나 많은 그리고/또는 어떤 타입의 트래픽이 새로운 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트 대비(versus) 구 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트에 디렉팅되어야 하는지를 제어하도록 구성될 수 있다.

[0085] 오직 두 개의 버전의 어플리케이션이 전개될 수 있는(그리고 하나의 버전의 어플리케이션이 철수(retirement)를 위해 표시될 필요가 있는) 어플리케이션의 프로덕션 재전개(production redeployment)와는 달리, 하나의 실시예에 따르면, 시스템은 두개 보다 많은 버전의 어플리케이션이 동시에 전개되고 활성화되게 할 수 있으며, 단 하나의 요건은 이들이 서로 다른 파티션들에 전개되어야 한다는 것이다.

[0086] 하나의 실시예에 따르면, 시스템은 또한, 클러스터 레벨에서 특정한 패치 레벨을 유지하되 예컨대 일부 파티션들이 특정한 시간에 특정한 패치 레벨을 지원할 수 없음이 결정되는 경우 필요에 따라 다양한 클러스트들에 상기 파티션들을 이동시켜, 기본 로직(underlying logic)을 공유하기 위한 복수의 테넌트들의 능력을 지원한다.

[0087] 마찬가지로, 하나의 실시예에 따르면, 시스템은, 테스트 목적으로 하나의 노드에서 예컨대, 오라클 홈의 패치 레벨 버전을 이용하고 그 다음, 일단 테스트가 완료되면 필요에 따라 그 버전의 오라클 홈을 다른 노드들에 롤 아웃시키기 위한 능력을 지원한다.

[0088] 도 8은 또한, 하나의 실시예에 따른 세션 호환성 검출에 대한 지원을 포함하는 패치를 위한 시스템을 도시한다.

[0089] 도 8에 도시된 바와 같이, 하나의 실시예 및 본 명세서에 예시된 예에 따르면, 클러스터(500)는, 패치된 서버들(502), 이용불가능한 서버들(504) 및 패치되지 않은 서버들(506)의 그룹들을 포함하는 복수의 그룹들에 제공되는 (MS1 내지 MS5로서 표시되는) 복수의 피관리 서버들을 포함할 수 있다.

[0090] 하나의 실시예에 따르면, 피관리 서버가 (MS3이 엑스 표시된 바와 같이) 이용불가능해질 때, 트래픽 디렉터(예컨대, OTD)는 MS3이 다운됨을 표시하는 에러 메시지(511)를 수신할 수 있다. 트래픽 디렉터는 다른 피관리 서버

(MS2)에 컨택하는 것을 시도(512)할 수 있는 바, 상기 다른 피관리 서버는 직렬화 해제 에러를 검출할 시 웹 컨테이너로 하여금 예컨대, 페일오버그룹 해더 정보를 갖는 503 메시지를 리턴하게 할 것이다. 트래픽 디렉터는 이번에는 피관리 서버(MS4)에 503 헤더 정보에 기초하여 요청을 재시도(513)할 수 있다. 그 다음, MS4에서의 어플리케이션 서버는 MS2로부터 적절한 세션 정보(514)를 패치할 수 있고, 마지막으로 상기 요청에 응답(515)한다.

[0091] 하나의 실시예에 따르면, 프로세스는 하기에 더 기술되는 바와 같이 지연 세션 직렬화 해제(518) 기능의 이용을 레버리지할 수 있다.

[0092] 도 9는 또한, 하나의 실시예에 따른 패치를 위한 시스템을 도시한다.

[0093] 도 9에 도시된 바와 같이, 하나의 실시예에 따르면, 시스템은 도메인 내의 클러스터가 다른 홈 디렉토리, 예컨대 다른 오라클 홈을 이용할 수 있게 하고 그러므로, 다른 어플리케이션 서버(예컨대, WLS) 버전 또는 패치 버전을 이용하여 동작할 수 있게 한다. 클러스터에 대한 피관리 서버들은 동일한 도메인으로부터의 다른 클러스터들을 지원하는 어떤 피관리 서버들과 동일한 또는 서로 다른 호스트들에 상주할 수 있다.

[0094] 특히, 도 9에 도시된 바와 같이, 시스템은 C1(530), C2(532) 및 C3(534)를 포함하는 복수의 클러스터들을 포함할 수 있고, 이들 각각은 파티션 A(562), 파티션 B(564), 파티션 C(566) 및 파티션 N(568)으로 표시된 하나 이상의 파티션들(550, 552, 554)을 동작시킨다.

[0095] 하나의 실시예에 따르면, 패치 데이터베이스(540)는 버전 A(542), 버전 B 패치 세트 1(PS1)(544) 및 버전 B 패치 세트 2(PS2)(546)로서 표시된 복수의 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트들에 대한 버전 또는 패치 정보를 포함할 수 있다.

[0096] 하나의 실시예에 따르면, 서로 다른 파티션들은 서로 다른 시간에 이주(migrate) 및/또는 패치될 수 있어서, 예컨대 파티션 A는 제1 버전 A의 특정한 어플리케이션 서버(예컨대, WLS)를 갖는 클러스터 C1으로부터 다른 버전 B PS1의 어플리케이션 서버를 갖는 클러스터 C2로 이주될 수 있다. 마찬가지로, 파티션 C는 버전 A의 어플리케이션 서버를 갖는 클러스터 C1으로부터 또다른 버전 B PS2의 어플리케이션 서버를 갖는 클러스터 C3로 이주될 수 있다.

[0097] 하나의 실시예에 따르면, 이 패치 프로세스의 일부 장점들은 개별 파티션들이 동일한 리소스들을 공유하는 다른 파티션들에 영향을 끼침이 없이 더 새로운(예컨대, 패치된) 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트(예컨대, 더 새로운 버전의 WLS)에 이주될 수 있게 하는 것을 포함한다. 패치 프로세스는 또한, 패치된 버전의 WLS 대비 초기 버전의 WLS 어플리케이션 서버의 A/B 테스트 또는 특정한 버전의 WLS를 갖는 서로 다른 버전의 어플리케이션의 테스트를 가능하게 한다.

[0098] 하나의 실시예에 따르면, 일정 시간 기간 동안, 파티션은 두 개의 클러스터들(예컨대, 소스 및 타겟 클러스터)에서 동시적으로 "라이브(live)"인 것으로서 고려될 수 있는 바, 이는 어떤 기존 세션들이 완료되거나 또는 타임아웃되게 할 수 있다. 일단, 파티션 이주가 완료되면, 상기 파티션은 어떤 새로운(예컨대, 패치된) 버전의 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트를 포함하는 타겟 클러스터에만 이용가능해지게될 것이다.

[0099] 도 10은 하나의 실시예에 따른 패치를 위한 시스템을 도시한다.

[0100] 도 10에 도시된 바와 같이, 하나의 실시예에 따르면, 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트가 실행되는 하나 이상의 컴퓨터 노드들 또는 서버들을 패치하기 위해, 이 노드들 상의 서버들은 먼저 정상적으로 셧다운된다.

[0101] 단계(580)에서, 준비 스위치(prepare switch)(예컨대, prepareSwitchOracleHome) 프리미티브가 패치될 노드 또는 서버에서 콜(call)되는 바, 이는 상기 노드 또는 서버에 대한 노드 관리자가 자신의 홈 디렉토리(예컨대, 오라클 홈)의 스위칭을 수행할 스크리트를 셋업하도록 디렉팅한다. 이 단계는 노드 관리자에게 상기 노드 관리자가 동작을 수행하기 위해 요하는 파라미터들을 제공하기 위해 이용된다.

[0102] 단계(582)에서, 재시작 노드 관리자(예컨대, RestartNodeManager) 프리미티브에 대한 콜이 이루어지게 하고, 이 콜은 그 노드에서 상기 노드 관리자가 스크립트(예컨대, switchOracleHome script)에 대한 제어를 전달하게 하는 바, 이는 또한, 현재의 홈 디렉토리(예컨대, 오라클 홈)를 특정 디렉토리 경로로 이동(583)시키고, 패치된 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트 이미지를 본래의 위치 내로 추출하고 그 다음, 노드 관리자를 다시 시작시킬 것이다.

[0103] 단계(584)에서, 어서트 스위치(assert switch)(예컨대, AssertSwitchOracleHome) 프리미티브가 실행되고, 이는 홈(예컨대, 오라클 홈) 디렉토리들의 스위치(585)가 성공적으로 완료됨을 확인(confirm)할 것이다.

[0104] 단계(588)에서, 시작 서버(예컨대, StartServers) 프리미티브가 각각의 노드 또는 서버에 대해 콜되고, 레디 앱 체크(예컨대, ReadyAppCheck)가 (레디 앱 체크가 구성되는 경우) 성공적으로 리턴될 때까지 완료되지 않을 것이다. 이는 워크플로우가 어떤 더 많은 노드들 또는 서버들을 셧다운시키기 전에 그 노드에 패치된 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트들 모두가 요청들을 서비스할 수 있고 제한된 다운타임을 지원하거나 다운타임이 없음(즉, 제로 다운타임)을 지원하게 할 것이다.

[0105] 도 11 내지 12는 또한, 하나의 실시예에 따른 패치를 위한 시스템을 도시한다.

[0106] 도 11 내지 12에 도시된 바와 같이, 예시적인 실시예에 따르면, 시스템은 (어드민 노드(600)으로서 표시된) 자신만의 머신 상에서 스스로 실행되는 어드민 서버와 함께, (컴퓨터 노드들 1 내지 3로서 표시된) 세 개의 물리적 머신들 또는 노드들에 걸쳐 실행되는 클러스터(604) 내의 복수의 피관리 서버들을 포함할 수 있다. 동일한 머신 상의 클러스터 내의 피관리 서버들의 각각의 쌍은 동일한 로컬 도메인 디렉토리 및 동일한 로컬 홈(예컨대, 오라클 홈) 디렉토리를 공유한다. 각각의 머신은 자신만의 노드 관리자를 포함한다.

[0107] 하나의 실시예에 따르면, 초기에, 어드민 서버 및 피관리 서버들은 본래의 홈 디렉토리(602, 606, 607, 608)를 이용한다. 패치 프로세스는, 패치된 버전을 각각의 피관리 서버에 카피하고 그 다음, (서비스 인터럽션이 없이) 어드민 서버에 롤아웃을 수행함(610)으로써 진행될 수 있다.

[0108] 하나의 실시예에 따르면, 피관리 서버들은, 심지어 일부 피관리 서버들이 일시적으로 셧다운되는 동안에도, 패치되는 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트의 페일오버를 제공할 수 있게 하기에 충분한 머신들에 걸쳐 충분하게 분배된다. 그 다음, 피관리 서버들은 패치되고, 패치된 공유 스토리지(616, 617, 618)를 포인팅하는 롤링 재시작이 수행된다. 프로세스는 결과적으로, 상태 복제로 인한 세션 손실들이 없게 하고 제한된 다운타임을 가능하게 하거나 다운타임이 없게(즉, 제로 다운타임이 가능하게) 한다.

[0109] 예시적인 실시예

[0110] 예시적인 실시예에 따르면, 아웃-오프-플레이스 패치는 인터럽션이 없이 동작하기 위한 도메인의 능력을 유지하는 제어된 롤링 재시작으로 패치들을 적용하기 위해 클러스터링에 구축된 기존 고 가용성 특징들의 장점을 취한다. 이 프로세스는, 복잡하고 장기간 실행되는 태스크들을 자동화하고, 롤백을 위해 패치되지 않은 (또는 이전) 버전을 보존하고, 복구불가능한 에러의 이벤트에서 자동 리버전을 제공함으로써 노출을 감소시키도록 설계된다. 상위 레벨에서, 상기 프로세스는, 도메인의 서버들에 의해 사용되는 오라클 홈 디렉토리 또는 디렉토리들을 클로닝(clone)하고, 제로 다운타임 호환가능 패치들을 복제 디렉토리들(duplicate directories)에 적용하고, 롤아웃을 처리할 오케스트레이션 태스크를 시작하는 것이다.

[0111] 하나의 실시예에 따르면, 롤아웃 태스크는 차례로 각각의 서버에 대해 다음을 코디네이션(coordination)할 것이다: 공통 도메인(디렉토리)을 공유하는 노드 상의 서버들을 정상적으로 셧다운, 서버와 관련된 노드 관리자를 재시작, 현재의 오라클 홈 디렉토리를 백업 위치로 이동시키고 특정 오라클 홈 디렉토리를 그 자리에 전개, 서버를 시작하고 그리고 구성된 경우 ReadyAppsCheck를 대기.

[0112] 일부 경우들에서, 서버들의 구성에 기초하여, 하나의 시간에 하나보다 많은 서버를 셧다운시키는 것이 바람직할 수 있다. 어떤 하나의 시간에 셧다운되는 서버들의 수는 롤아웃의 효과를 최소화하기 위해 가능한 작게 유지되어야 한다. 클러스터에는 업 상태(up)이고 요청들에 응답할 수 있는 적어도 하나의 서버가 항상 존재할 것이다.

[0113] 복구불가능한 에러의 이벤트 시, 롤아웃 태스크는 자신이 행한 어떤 변화를 자동으로 복귀시켜서(revert), 서버들이 자신의 본래의 상태(이전의 버전)으로 리턴될 것이다. 이는 에러가 진단 및 해결되는 동안 도메인이 완전히 이용가능하게 할 것이다. 롤백은 본래의 오라클 홈 디렉토리를 보존함으로써 가능해지고, 패치들이 본래의 디렉토리 대신 복제 디렉토리에 적용되는 이유 중 일부이다. 롤백 프로세스 동안 롤백이 완료되는 것을 금지시키는 다른 에러에 직면하면, 에러가 발생될 것이고, 프로세스는 조사(investigation)를 할 수 있도록 정지될 것이다. 일단 에러가 해결되면, 복귀 프로세스가 재개될 수 있다.

[0114] 초기 인스톨레이션 및 구성(Initial Installation and Configuration)

[0115] 하나의 실시예에 따르면, 아웃-오브-플레이스 패치를 지원하기 위해, 충족되어야 하는 여러 요건들이 서버들에 걸친 어플리케이션 서버(예컨대, WLS)의 인스톨레이션에 대해 존재한다.

[0116] 도메인에는 오라클 홈의 위치가 참조되는 많은 장소들이 존재한다. 이는 시작 스크립트들, 속성 파일들 및 xml 구성 파일들 내의 변수들(variables)을 포함한다. 일반적으로, 새로운 버전의 오라클 홈을 포인팅하기 위해 위치들의 모두를 발견하고 업데이트하는 것은 실용적이지 않다. 이러한 이유로, 하나의 실시예에 따르면, 롤아웃은, 기존 오라클 홈을 (사용자가 특정한 백업 위치에) 이동시키고 그리고 요구되는 오라클 홈을 그 자리에 확장시킴으로써 작용한다. 이러한 절차가 여전히 동작중인 피관리 서버들에 영향을 끼치지 않도록 하기 위해, 오라클 홈 디렉토리는 머신 상의 영향을 받는 피관리 서버들의 전부에 의해 이용되어야 하고 다른 머신들 상의 피관리 서버들에 의해서는 이용되지 않아야 한다. 오라클 홈은 또한, 노드 관리자 프로세스에 의해 기록가능(writable)한 위치에 있어야 한다. 이러한 조건들을 보장하기 위해, 오라클 홈 디렉토리는 영향을 받는 피관리 서버들에 로컬한 하드 드라이브 상에 인스톨될 수 있다.

[0117] 서버를 업그레이드하는 동안 업타임(uptime)을 유지하기 위한 핵심(key)은 클러스터들로 구성된 고가용성의 장점을 취하는 것이다. 클러스터 내의 최소 개수의 서버들이 모든 시간에 작동 상태(operational)를 유지해야 한다. 동일한 머신 상의 클러스터 내의 서버들이 (이들이 공통 도메인 디렉토리를 공유하는 경우) 함께 재시작될 필요가 있기 때문에, 클러스터 내의 서버들은 적어도 두 개의 서로 다른 물리적 머신들 상에서 호스팅됨이 요구되되, 클러스터 당 최소 세 개의 머신들이 권고된다. 이는 일부가 깨어 있는 상태로 유지(stay up)되게 하고 나머지는 롤링 재시작의 일부로서 다운되게 된다.

[0118] 요청을 처리하기 위해 서로 다른 머신들 상에서 이용가능한 서버들의 수를 결정할 때, 실행중이되 어드민 또는 스탠바이 모드에 있는 피관리 서버들을 배제시키는 것은 중요한 바, 이 서버들이 요청들에 응답하지 않을 것이기 때문이다.

[0119] 롤아웃 프로세스는 어드민 서버 및 피관리 서버가 동시에 업데이트될 필요가 있는 경우 매우 복잡해질 수 있다. 이는 어드민 서버와 피관리 서버가 동일한 머신 상에서 실행되고 동일한 도메인 디렉토리를 공유하도록 구성되는 경우일 수 있다. 어드민 서버는 공유되는 오라클 홈 디렉토리로부터 실행 중일 수 있기 때문에 피관리 서버와 동시에 다운되게 할 필요가 있다. 이러한 제약은 피관리 서버별로 패치들을 롤아웃 할 수 있도록 피관리 서버의 인스톨레이션 홈이 격리되는 경우 적용되지 않을 것이다. 이러한 이유로 이 문제를 단순화시키는 두 가지 서로 다른 구성들이 지원된다.

[0120] 1. 첫 번째는 어떤 피관리 서버들이 머신 상에서 실행됨이 없이 어드민 서버가 상기 머신 상에서 실행되게 하는 것이다. 이는 어드민 서버가 스스로 하나의 단계에서 업데이트될 수 있게 하며, 이것이 일단 완료되면, 다음 단계는 다른 머신들 상의 해당 도메인의 피관리 서버를 업데이트하는 것일 것이다.

[0121] 2. 두 번째 구성은, 어드민 서버가 피관리 서버와 동일한 머신 상에서 실행될 수 있게 하되, 자신만의 별개의 도메인 디렉토리를 소진(run out of)하게 하는 것이다. 이는 다시, 어드민 서버가 개별적으로 업데이트될 수 있게 하며, 피관리 서버는 자체 단계에서 업데이트될 수 있다.

[0122] 도메인 내의 모든 서버들을 업데이트할 메커니즘을 제공하는 것에 추가적으로, 이 특징은 또한, 도메인 내의 개별적인 클러스터들을 업데이트하기 위한 능력을 제공한다. 사용자가 클러스터 롤아웃 모드를 이용하는 것을 시도할 때, 서로 다른 클러스터들을 서빙하는 단일 노드 상에 복수의 피관리 서버들이 존재하는 경우, 피관리 서버들은 이들이 서빙하는 클러스터에 따라 개별 도메인 디렉토리들을 가져야 한다. 이들의 도메인 디렉토리들은 또한, 개별적인 오라클 홈 디렉토리들을 포인팅해야 하며, 이들은 마찬가지로, 노드 관리자의 개별 인스턴스에 의해 관리되어야 한다. 이는, 클러스터에 대한 노드 상의 피관리 서버들 모두가 다운되게 되고 다른 클러스터를 서빙하고 (여전히 실행중인) 피관리 서버들의 오라클 홈 디렉토리에 영향을 끼침이 없이 자신의 오라클 홈 디렉토리를 업데이트되게 하도록 요구된다.

[0123] 도메인 내에서 서로 다른 시간에 서로 다른 파티션들을 패치하는 것은 명시적으로(explicitly) 지원되지 않지만, 파티션들을 관리하고 클러스터 레벨 패치를 이용함으로써 달성하는 것이 가능하다. 파티션들이 그 환경에서 어떻게 이용되는지에 따라, 하나의 파티션을 업그레이드함이 없이 다른 파티션을 업그레이드하는 것이 바람직할 수 있다. 이의 예는, 각각의 파티션이 서로 다른 테넌트에 의해 이용되고 하나의 테넌트가 업그레이드할 필요가 있되 다른 테넌트는 이용가능한 유지관리 윈도우(maintenance window)를 가지지 않는 환경일 수 있다. 이러한 상황에서, 파티션 이주 특징은 파티션들을 분리하기 위해 이용될 수 있다. 업그레이드가 필요한 파티션은 (기존 또는 새롭게 생성된) 다른 클러스터에 이주될 수 있고, 클러스터 레벨 롤아웃은 새로운 클러스터 상에서 수행될 수 있다. 이를 달성하기 위한 가장 단순한 방식은 새로운 클러스터가 본래의 클러스터와 다른 물리적 머신들 상에 호스팅되는 경우인 바, 이는 도메인 디렉토리, 오라클 홈 및 노드 관리자가 오버랩되지 않게 할 것이다. 다른 물리적 리소스들이 이용가능하지 않은 경우, 이 절차는, 새로운 클러스터가 오라클 홈 디렉토리의

자신만의 카피를 포인팅하는 도메인 디렉토리의 자신만의 카피를 가지며 각각의 영향을 받는 머신 상에서 실행되는 노드 관리자의 자신만의 인스턴스를 가지는 한은 여전히 지원될 수 있다.

[0124] 하나의 실시예에 따르면, 노드 관리자는 현재의 오라클 홈을 특정된 백업 디렉토리에 이동시키고 새로운 오라클 홈을 추출 또는 그 자리에 카피하는 것을 담당한다. 노드 관리자는 또한, 새로운 디렉토리를 소진하기 위해 재시작되어야 한다. 이를 코디네이션하기 위해 각각의 노드는 자신만의 노드 관리자를 가져야한다.

[0125] 예를 들어, 상기 기술된 특징들 10 내지 12에서, 시스템은 자신 만의 머신 상에서 스스로 실행되는 어드민 서버와 함께, 세 개의 물리적 머신들에 걸쳐 실행되는 클러스터 내의 복수의 피관리 서버들을 포함한다. 동일한 머신 상의 클러스터 내의 피관리 서버들의 각각의 쌍은 동일한 로컬 도메인 디렉토리 및 동일한 로컬 오라클 홈 디렉토리를 공유하고, 각각의 머신은 자신만의 노드 관리자가 실행되게 한다.

[0126] 클로닝 및 클로닝된 이미지 패치

[0127] 하나의 실시예에 따르면, 기존 이미지를 클로닝하고 클로닝된 이미지를 패치하기 위해, 시스템은 기존 툴, 예컨대 기존 오라클 홈을 클로닝하기 위한 FMW 이동 스크립트들의 이용에 의존할 수 있다. 일단, 클로닝된 오라클 홈이 존재하면, 사용자는 이미지를 패치하기 위해 기존 OPatch 툴을 이용할 수 있다. FMW 이동 스크립트들로 오라클 홈을 클로닝하는 것이 아래에 설명된다.

[0128] 1. WLS 인스톨의 아카이브를 만들기 위해 copyBinary.sh을 이용.

[0129] 2. WLS 인스톨의 클론을 만들기 위해 새로운 디렉토리에 pasteBinary.sh를 이용(중앙 인벤토리 파일을 업데이트). 일단, 클론이 생성되면, 상기 이용은 오라클 유니버셜 인스톨러를 실행시키고, 클론이 중앙 인벤토리에 등록됨을 알 수 있다.

[0130] 자동화된 롤아웃

[0131] 상기 기술된 바와 같이, 하나의 실시예에 따르면, 제로 다운타임으로 업데이트를 롤 아웃하는 것은 대부분, 서버 클러스터링의 고가용성 특징들을 레버리지함으로써 가능해진다. 서버 클러스터링의 경우, 피관리 서버들 중 하나 이상은 어플리케이션이 다운타임을 겪게 함이 없이 오프라인일 수 있다. 사실 상, 정상적인 서버 셧다운의 경우, 심지어 단일 세션이 손실되는 것을 방지하는 것이 대부분의 경우에 가능하다. 서버들을 다운시키고(take down), 이들을 업데이트하고 그리고 이들을 서비스에 다시 가져오는 것의 코디네이션은, 패치 프리미티브들로서 지칭되는 커스텀 커맨드들을 생성하고 그리고 이들을 오케스트레이션 프레임워크를 이용하여 실행함으로써 처리될 수 있다. 상기 커맨드들은 도메인의 토폴로지를 분석하고, 서버들 및 노드 관리자들을 차례대로 안전하게 업데이트하기 위한 가장 양호한 방식을 결정하며, 오케스트레이션 프레임워크는 프로세스의 모니터링 및 에러 처리를 제공한다.

[0132] 하나의 실시예에 따르면, 이 메커니즘이 적적하게 작용하게 하기 위해, 업그레이드되는 클러스터 내의 피관리 서버들은 둘 이상의 물리적 머신들에 걸쳐 스프레드되어야 한다. 이는 동일한 머신에 의해 호스트되는 클러스터 내의 모든 서버들이 공통 도메인 디렉토리를 공유하고 따라서, 함께 다운되어야(bring down) 하기 때문이다. 다운타임을 회피하기 위해, 클러스터 내의 서버들 중 일부는 다른 서버들과 다른 머신 상에서 실행되어야 한다. 이 방식은 서비스 요청들에 이용가능한 일부 서버들이 항상 존재한다.

[0133] 이 기법에 의해 도입되는 다른 제약은, 클로닝된 오라클 홈에 적용되는 패치들은 이 서버들을 패치되지 않은 서버들과 여전히 호환가능한 상태로 두어야 한다는 요건이다. 특히, 패치 롤아웃 동안의 서버 고장(failing)의 이벤트 시, 사용자의 세션이 패치된 서버와 패치되지 않은 서버 간에 매끄럽게(seamlessly) 이주되는 것이 가능해야 한다.

[0134] 하나의 실시예에 따르면, 이러한 방식으로 롤 아웃될 수 있는 여러 동작들이 존재한다. 이들은 패치된 오라클 홈을 롤아웃시키는 것, 서버들에 걸쳐 JAVAJHOME의 위치를 업데이트하는 것, 어플리케이션들을 업데이트된 버전들과 교체하는 것 그리고 단일 롤아웃에서의 이러한 동작들의 어떤 조합을 포함한다. 모든 서버들에 걸쳐 롤링 재시작을 수행하기 위한 능력이 또한 제공된다.

[0135] 예시적인 패치 API들

[0136] 하나의 실시예에 따르면, 업그레이드들 또는 패치들을 롤아웃하기 위해 이용될 수 있는 예시적인 패치 API들이 하기에 기술된다. 다른 실시예들에 따르면, 다른 그리고/또는 추가적인 패치 API가 지원될 수 있다.

[0137] rolloutUpdate(target, [rolloutOracleHome, backupOracleHome, isRollback], [javaHome],

[applicationProperties], [options])

[0138] rolloutJavaHome(target, javaHome, [options])

[0139] rolloutApplications(target, applicationProperties, [options])

[0140] rolloutOracleHome(target, rolloutOracleHome, backupOracleHome, isRollback, [options])

[0141] rollingRestart(target)

[0142] RolloutUpdate 커맨드

[0143] 하나의 실시예에 따르면, RolloutUpdate 커맨드는 오라클 홈, 자바 홈 및 서버들 상의 어플리케이션들을 업데이트하기 위한 능력을 제공한다. 또한, 이는 옵션적 파라미터들 중 어느 것이 특정되는지에 따라 이 변경들의 모든 조합을 가능하게 한다. 오라클 홈을 업데이트하기 위해, 사용자는 rolloutOracleHome, backupOracleHome 및 isRollback 파라미터들을 특정해야 한다. 자바홈을 업데이트하기 위해, 사용자는 javaHome 파라미터를 특정해야 한다. 어플리케이션들을 업데이트하기 위해, 사용자는 applicationProperties 파라미터를 특정해야 한다. isDryRun 및 autoRevertOnFailure 옵션들은 모든 경우에 유효하며 isSessionCompatible 옵션은 어플리케이션들 및/또는 오라클 홈이 수정되는 경우에만 고려될 것이다. 업데이트들이 단일 롤아웃 중에 수행될 수 있는 제약이 존재하지 않는다. 사용자가 Oracle Home 파리미터들, JavaHome 파라미터 또는 ApplicationProperties 파라미터를 특정하지 않으면, 롤링 재시작이 수행된다.

[0144] 신택스(syntax)

[0145] rolloutUpdate(target, [rolloutOracleHome, backupOracleHome, isRollback], [javaHome], [applicationProperties], [options])

표 1

[0146]

인수(Argument)	정의
target	서버들의 도메인, 클러스터 또는 이름에 기초하여 어느 서버들이 영향을 받을지를 특정하는 방식 도메인의 이름 - 변경들은 도메인 내의 모든 서버들에 적용될 것이다 클러스터의 이름 또는 클러스터들의 쉼표 구분 리스트 - 변경들은 특정된 클러스터들 중 하나에 속하는 모든 서버들에 적용될 것이다 서버의 이름 또는 서버들의 쉼표 구분 리스트 - 변경들은 모든 특정된 서버들에 적용될 것이다
rolloutOracleHome	현재의 오라클 홈과 대체될 롤아웃을 위한 오라클 홈의 버전을 포함하는 아카이브 또는 로컬 디렉토리의 위치 아카이브는 전형적으로 copyBinary 스크립트로 생성되는 jar 파일이다
backupOracleHome	현재의 오라클 홈이 이동/재명명될 로컬 디렉토리의 이름
isRollback	사용자가 도메인에 롤아웃되는 변경이 오라클 홈의 이전 패치 릴리즈에 이루어짐을 특정할 수 있게 함 이 정보는 어드민 서버가 처음 또는 마지막으로 업데이트되어야 하는지를 결정하는 데 있어 중요하다 타겟이 도메인이고 롤아웃되는 오라클 홈이 현재의 오라클 홈보다 낮은 패치 버전을 가지면 TRUE, 그렇지 않으면 FALSE
javaHome	이용할 새로운 JAVA_HOME의 위치. 새로운 JAVA_HOME은 각각의 머신 상에 인스톨되는 유효한 JAVA_HOME 경로를 참조해야한다. 롤아웃 동작은 Java 인스톨러를 실행시키지 않을 것이다.

applicationProperties	업그레이드될 각각의 어플리케이션에 관한 정보를 포함하는 어드민 서버 상에서 판독가능한 파일의 위치를 지정하는 데 사용되며, 상기 파일은 하기 예시된 바와 같이 JSON 포맷으로 특정되는 어플리케이션 정보를 갖는 텍스트 파일이다:
	{"applications":[
	{
	"applicationName":"App1",
	"patchedLocation":"/pathto/patchedLocation1",
	"backupLocation":"/pathto/backupLocation1"
	},
	{
	"applicationName":"App2",
	"patchedLocation":"/pathto/patchedLocation2",
	"backupLocation":"/pathto/backupLocation2"
	},
	{
	"applicationName":"App3",
	"patchedLocation":"/pathto/patchedLocation3",
	"backupLocation":"/pathto/backupLocation3"
	}
	]}

options	옵션적임. 이름-값(name-value) 쌍들로서 특정된 롤아웃 옵션들의 쉼표 분리 리스트. 유효한 옵션들은 다음을 포함한다: 동작이 평가되되 실행되지 않는 경우 isDryRun = TRUE, 디폴트는 FALSE 동작이 실패시 자동으로 복귀되어야 하는 경우 autoRevertOnFailure = TRUE (디폴트), 동작이 실패시 정지되어 사용자가 이를 재개하길 대기해야 하는 경우 FALSE 오라클 홈의 패치된 버전과 패치되지 않은 버전 간에 세션들이 호환가능한(세션 처리 및 정상적인 서버 셧다운 시간들에 영향을 끼치는) 경우 isSessionCompatible = TRUE, 패치되지 않은 세션들을 보존하는 것에 대한 특별한 고려가 이루어져야 하는 (롤아웃이 완료되는 데 소요되는 시간에 영향을 끼칠 수 있는) 경우 FALSE (디폴트)

[0147] 예시

[0148] 새로운 패치된 오라클 홈을 롤아웃:

[0149] > progress = rolloutUpdate(DomainA, /pathto/wls1221p2.jar, /pathto/backupOfwls1221p1, FALSE)

[0150] 본래의 오라클 홈으로 롤백

[0151] > progress = rolloutUpdate(DomainA, /pathto/backupOfwls1221p1, /pathto/backupOfwls1221p2-broken, TRUE)

[0152] 새로운 버전의 자바만을 롤아웃

[0153] > progress = rolloutUpdate(DomainA, javaHome=/pathto/jdk1.8.0_55)

[0154] 업그레이드된 어플리케이션들만을 롤아웃

[0155] > progress = rolloutUpdate(DomainA, applicationProperties=/pathto/applicationProperties)

[0156] 새로운 버전의 자바를 갖는 새로운 패치된 오라클 홈을 롤아웃

[0157] > progress = rolloutUpdate(DomainA, /pathto/wls1221p2.jar, /pathto/backupOfwls1221p1, FALSE, /pathto/jdk1.8.0_55)

[0158] 새로운 패치된 오라클 홈, 새로운 버전의 자바 및 업그레이드된 어플리케이션들을 롤아웃

[0159] > progress = rolloutUpdate(DomainA, /pathto/wls1221p2.jar, /pathto/backupOfwls1221p1, FALSE, /pathto/jdk1.8.0_55, /pathto/applicationProperties)

[0160] RolloutOracleHome 커맨드

[0161] 하나의 실시예에 따르면, rolloutOracleHome 커맨드는 오라클 홈을 업데이트하기 위한 능력을 제공한다. rolloutOracleHome 태스크는 어느 서버들이 어떤 순서로 업데이트될 필요가 있는지를 알아내고 이 서버들을 안전하게 업데이트할 워크플로우를 생성하는 것을 담당한다. 이는 서버들의 정상적인 셧다운, 오라클 홈 디렉토리를 교체하는 것, 노드 관리자를 재시작하는 것 및 서버들을 다시 시작하는 것을 포함한다. 롤아웃 태스크는 상태에 대해 폴링될 수 있는 WorkflowProgressMBean을 리턴할 것이다.

[0162] 신텍스

[0163] rolloutOracleHome(target, rolloutOracleHome, backupOracleHome, isRollback, [options])

표 2

[0164]

인수	정의
target	서버들의 도메인, 클러스터 또는 이름에 기초하여 어느 서버들이 영향을 받을지를 특정하는 방식 도메인의 이름 - 변경들은 도메인 내의 모든 서버들에 적용될 것이다 클러스터의 이름 또는 클러스터들의 쉼표 구분 리스트 - 변경들은 특정된 클러스터들 중 하나에 속하는 모든 서버들에 적용될 것이다 서버의 이름 또는 서버들의 쉼표 구분 리스트 - 변경들은 모든 특정된 서버들에 적용될 것이다
rolloutOracleHome	현재의 오라클 홈과 교체할 롤아웃을 위한 오라클 홈의 버전을 포함하는 아카이브 또는 로컬 디렉토리의 위치 아카이브는 전형적으로 copyBinary 스크립트로 생성되는 jar 파일이다.
backupOracleHome	현재의 오라클 홈이 이동/재명명될 로컬 디렉토리의 이름
isRollback	사용자가 도메인에 롤아웃되는 변경이 오라클 홈의 이전 패치 릴리즈에 이루어짐을 특정할 수 있게 함. 이 정보는 어드민 서버가 처음 또는 마지막으로 업데이트되어야 하는지를 결정하는 데 있어 중요하다 타겟이 도메인이고 롤아웃되는 오라클 홈이 현재의 오라클 홈보다 낮은 패치 버전을 가지면 TRUE, 그렇지 않으면 FALSE
options	옵션적임. 이름-값 쌍들로서 특정된 롤아웃 옵션들의 쉼표 분리 리스트. 유효한 옵션들은 다음을 포함한다: 동작이 평가되되 실행되지 않는 경우 isDryRun = TRUE, 디폴트는 FALSE 동작이 실패시 자동으로 복귀되어야 하는 경우 autoRevertOnFailure = TRUE (디폴트), 동작이 실패시 정지되어 사용자가 이를 재개하길 대기해야 하는 경우 FALSE 오라클 홈의 패치된 버전과 패치되지 않은 버전 간에 세션들이 호환가능한(세션 처리 및 정상적인 서버 셧다운 시간들에 영향을 끼치는) 경우 isSessionCompatible = TRUE, 패치되지 않은 세션들을 보존하는 것에 대한 특별한 고려가 이루어져야 하는(롤아웃이 완료되는 데 소요되는 시간에 영향을 끼칠 수 있는) 경우 FALSE (디폴트)

[0165] 예시

[0166] 패치된 오라클 홈을 롤아웃

[0167] > progress = rolloutOracleHome(DomainA, /pathto/wls1221p2.jar, /pathto/backupOfwls1221p1, FALSE)

[0168] RolloutJavaHome 커맨드

[0169] 하나의 실시예에 따르면, rolloutJavaHome 커맨드는 영향을 받는 서버들에 의해 이용되는 자바홈을 업데이트하기 위한 능력을 제공한다. rolloutJavaHome 태스크는 어느 서버들이 어떤 순서로 업데이트될 필요가 있는지를 알아내고 이 서버들을 안전하게 업데이트할 워크플로우를 생성하는 것을 담당한다. 이는 서버들의 정상적인 셧다운, 이들이 사용하는 자바홈의 위치를 업데이트하는 것, 노드 관리자를 재시작하는 것 및 서버들을 다시 시작하는 것을 포함한다. 이 태스크는 상태에 대해 폴링될 수 있는 WorkflowProgressMBean을 리턴할 것이다.

[0170] 신텍스

[0171] rolloutJavaHome(target, javaHome, [options])

표 3

[0172]

인수	정의
target	서버들의 도메인, 클러스터 또는 이름에 기초하여 어느 서버들이 영향을 받을지를 특정하는 방식 도메인의 이름 - 변경들은 도메인 내의 모든 서버들에 적용될 것이다 클러스터의 이름 또는 클러스터들의 쉼표 구분 리스트 - 변경들은 특정된 클러스터들 중 하나에 속하는 모든 서버들에 적용될 것이다 서버의 이름 또는 서버들의 쉼표 구분 리스트 - 변경들은 모든 특정된 서버들에 적용될 것이다
javaHome	사용할 JAVA_HOME의 위치. 새로운 JAVA_HOME은 각각의 머신 상에 인스톨되는 유효한 JAVA_HOME 경로를 나타내야 한다. 롤아웃 동작은 JAVA 인스톨러를 실행시키지 않을 것이다.
options	옵션적임. 이름-값 쌍들로서 특정된 롤아웃 옵션들의 쉼표 분리 리스트. 유효한 옵션들은 다음을 포함한다: 동작이 평가되되 실행되지 않는 경우 isDryRun = TRUE, 디폴트는 FALSE 동작이 실패시 자동으로 복귀되어야 하는 경우 autoRevertOnFailure = TRUE (디폴트), 동작이 실패시 정지되어 사용자가 이를 재개하길 대기해야 하는 경우 FALSE

[0173] 예시

[0174] 자바의 최근 인스톨된 버전을 이용하기 위해 도메인 내의 모든 서버들 상의 자바홈을 업데이트

[0175] > progress = rolloutJavaHome(DomainA, /pathto/jdk1.8.0_55)

[0176] RolloutApplications 커맨드

[0177] 하나의 실시예에 따르면, rolloutApplications 커맨드는 서버들 상에 전개되는 어플리케이션들을 업데이트하기 위한 능력을 제공한다. rolloutApplications 태스크는 어느 서버들이 어떤 순서로 업데이트될 필요가 있는지를 알아내고 이 서버들을 안전하게 업데이트할 워크플로우를 생성하는 것을 담당한다. 이는 서버들의 정상적인 셧다운, 어플리케이션들을 업데이트하는 것, 노드 관리자를 재시작하는 것 및 서버들을 다시 시작하는 것을 포함한다. 이 태스크는 상태에 대해 폴링될 수 있는 WorkflowProgressMBean을 리턴할 것이다.

[0178] 신텍스

[0179] rolloutApplications(target, applicationProperties, [options])

표 4

[0180]

인수	정의
target	서버들의 도메인, 클러스터 또는 이름에 기초하여 어느 서버들이 영향을 받을지를 특정하는 방식 도메인의 이름 - 변경들은 도메인 내의 모든 서버들에 적용될 것이다 클러스터의 이름 또는 클러스터들의 쉼표 구분 리스트 - 변경들은 특정된 클러스터들 중 하나에 속하는 모든 서버들에 적용될 것이다 서버의 이름 또는 서버들의 쉼표 구분 리스트 - 변경들은 모든 특정된 서버들에 적용될 것이다

applicationProperties	업그레이드될 각각의 어플리케이션에 관한 정보를 포함하는 어드민 서버 상에서 판독가능한 파일의 위치를 지정하는 데 사용되며, 상기 파일은 하기 예시된 바와 같이 JSON 포맷으로 특정되는 어플리케이션 정보를 갖는 텍스트 파일이다:
	
	"applications":[
	{
	"applicationName":"App1",
	"patchedLocation":"/pathto/patchedLocation1",
	"backupLocation":"/pathto/backupLocation1"
	},
	{
	"applicationName":"App2",
	"patchedLocation":"/pathto/patchedLocation2",
	"backupLocation":"/pathto/backupLocation2"
	},
	{
	"applicationName":"App3",
	"patchedLocation":"/pathto/patchedLocation3",
	"backupLocation":"/pathto/backupLocation3"
	}
	]

options	옵션적임. 이름-값 쌍들로서 특정된 롤아웃 옵션들의 쉼표 분리 리스트. 유효한 옵션들은 다음을 포함한다: 동작이 평가되되 실행되지 않는 경우 isDryRun = TRUE, 디폴트는 FALSE 동작이 실패시 자동으로 복귀되어야 하는 경우 autoRevertOnFailure = TRUE (디폴트), 동작이 실패시 정지되어 사용자가 이를 재개하길 대기해야 하는 경우 FALSE 오라클 홈의 패치된 버전과 패치되지 않은 버전 간에 세션들이 호환가능한(세션 처리 및 정상적인 서버 셧다운 시간들에 영향을 끼치는) 경우 isSessionCompatible = TRUE, 패치되지 않은 세션들을 보존하는 것에 대한 특별한 고려가 이루어져야 하는(롤아웃이 완료되는 데 소요되는 시간에 영향을 끼칠 수 있는) 경우 FALSE (디폴트)

[0181] 예시

[0182] 업그레이드된 어플리케이션들을 롤아웃

[0183] > progress = rolloutApplications(DomainA, /pathto/applicationProperties)

[0184] RollingRestart 커맨드

[0185] 하나의 실시예에 따르면, rollingRestart 커맨드는 서버들을 순차적으로 재시작하기 위한 능력을 제공한다. rollingRestart 태스크는 어느 서버들이 어떤 순서로 업데이트될 필요가 있는지를 알아내고 이 서버들을 안전하게 업데이트할 워크플로우를 생성하는 것을 담당한다. 이는 서버들의 정상적인 셧다운, 어플리케이션들을 업데이트하는 것, 노드 관리자를 재시작하는 것 및 서버들을 다시 시작하는 것을 포함한다. 이 태스크는 상태에 대해 폴링될 수 있는 WorkflowProgressMBean을 리턴할 것이다.

[0186] 신택스

[0187] rollingRestart(target, [options])

표 5

[0188]

인수	정의
target	서버들의 도메인, 클러스터 또는 이름에 기초하여 어느 서버들이 영향을 받을지를 특정하는 방식 도메인의 이름 - 변경들은 도메인 내의 모든 서버들에 적용될 것이다 클러스터 또는 클러스터들의 쉼표 구분 리스트의 이름 - 변경들은 특정된 클러스터들 중 하나에 속하는 모든 서버들에 적용될 것이다 서버 또는 서버들의 쉼표 구분 리스트의 이름 - 변경들은 모든 특정된 서버들에 적용될 것이다
options	옵션적임. 이름-값 쌍들로서 특정된 롤아웃 옵션들의 쉼표 분리 리스트. 유효한 옵션들은 다음을 포함한다: 동작이 평가되되 실행되지 않는 경우 isDryRun = TRUE, 디폴트는 FALSE 동작이 실패시 자동으로 복귀되어야 하는 경우 autoRevertOnFailure = TRUE (디폴트), 동작이 실패시 정지되어 사용자가 이를 재개하길 대기해야 하는 경우 FALSE

[0189] 예시

[0190] 도메인 내의 모든 서버들의 롤링 재시작을 행함

[0191] > progress = rollingRestart(DomainA)

[0192] 자바 홈 업데이트

[0193] 하나의 실시예에 따르면, 제로 다운타임 패치 특징은 특정된 타겟 내의 서버들에 대한 JAVA_HOME 설정을 업데이트하기 위한 메커니즘을 제공한다. 이 프로세스를 개시하기 위한 두 가지 방식들이 존재하는 바, 하나는 단독 커맨드 rolloutJavaHome를 이용하는 것이고, 나머지 하나는 rolloutUpdate 커맨드에 대한 옵션적인 JavaHome 파라미터를 특정함으로써 이루어진다. 후자를 이용할 때, 동일한 롤아웃에서 오라클 홈 및/또는 어플리케이션들을 업데이트하는 것이 가능하다. JAVA_HOME을 설정하는 기능은 마찬가지로 오라클 홈 또는 어플리케이션들이 업그레이드되는지에 관계없이 동일하다.

[0194] 하나의 실시예에 따르면, 오라클 홈을 업데이트하기 위한 상기 기술된 토폴로지 전제조건들(prerequisites)은 또한, 자바홈을 업데이트하는 것에 적용된다. 추가적으로, 이 기능을 제공할 수 있게 하기 위해, 포인팅할 JAVA_HOME을 설정하기 위한 자바의 버전이 로컬하게 액세스가능한 어떤 곳에 이미 인스톨되어야 함과 JAVA_HOME으로의 경로가 모든 영향을 받는 서버들에 대해 동일해야함이 요구된다. 서버들을 셧다운하기 전에 자바를 인스톨하는 것은 각각의 버전의 자바(현재 및 새로운 버전들)가 이들로의 별개의 고유한 경로를 가져야 함을 의미한다.

[0195] 하나의 실시예에 따르면, 자바홈에 대한 변경을 롤아웃하기 위해, 동일한 오라클 홈을 공유하는 머신 상의 모든 서버들은 그 머신 상에서 실행되는 노드 관리자와 더불어, 함께 셧다운되어야 한다. 이들이 셧다운되는 동안, 네이티브 스크립트(native script)는 새로운 JAVA_HOME 위치를 이용하기 위해 오라클 홈 디렉토리 내의 스크립트들 모두를 업데이트하기 위하여 pasteBinary의 특별한 형태를 이용할 것이다. 그 다음, 자바 업데이트 스크립트는 JAVA_HOME에 대한 새로운 경로를 또한 이용하기 위해 도메인 디렉토리 내의 필수적인(requisite) 시작 스크립트들을 수정할 것이다. 그 다음, 해당 머신 상의 노드 관리자 및 서버들이 다시 시작될 것이다. JAVA_HOME에 대한 참조를 포함하는 오라클 홈 하의 모든 스크립트들은 특정된 JAVA_HOME을 포인팅할 것이다. JAVA_HOME에 대한 참조를 포함하는 현재의 도메인 디렉토리 하의 모든 스크립트들은 특정된 JAVA_HOME을 포인팅할 것이다.

[0196] 성공적으로 수행된 자바홈에 대한 변경을 롤백하는 가장 쉬운 방식은 단순히, 구 위치(old location)를 새로운 경로로서 이용하여 새로운 updateJavaHome 커맨드를 실행하는 것이다. 그러나 어떤 인스턴스들에서, 시스템은 오라클 홈 변경 - 이는 자바홈 역시 변경했음 - 을 롤백하는 것을 지원한다. 오라클 홈 스크립트를 원래 상태로 리턴하는 것은 오라클 홈 디렉토리를 이전 상태로 복원하는 것의 고유한 부분으로 발생한다. 사용자가 롤백 커맨드를 발행할 때 본래의 (원하는) 자바홈 위치를 특정하지 않을 수 있기 때문에 도메인 스크립트들을 롤백하는 것은 간단하지 않을 수 있다. 이 문제를 해결하기 위해 updateOracleHome 커맨드를 적용하여 오라클 홈 디렉토리가 백업 위치로 이동될 때 이는 "domainBackup"이라는 추가 디렉토리를 포함하게 되며, 상기 추가적인 디렉토리는 업데이트의 시간에 현재의 버전의 관련 도메인 스크립트들의 카피를 유지할 것이다. 이 방식에서, 사용자가 예의 백업 오라클 홈 위치로부터 향후에 롤백 커맨드를 수행하는 경우, 이 도메인 파일들은 제자리에 다시

카피될 수 있다.

[0197] 어플리케이션들을 업데이트

[0198] 상기 기술된 바와 같이, 하나의 실시예에 따르면, 제로 다운타임 패치 특징은 마찬가지로 어플리케이션 서버들에 전개되는 어플리케이션들을 업데이트하기 위한 메커니즘을 제공한다. 이를 위한 한가지 메커니즘은 이들을 오라클 홈 디렉토리에 포함시키고 노-스테이지(no-stage)가 이들을 그곳으로부터 전개하는 것이다. 이러한 방식으로 전개되는 어플리케이션들을 업데이트하는 것은 (업데이트된 어플리케이션이 포함된) 새로운 버전의 오라클 홈이 롤아웃될 때 발생한다. 이러한 방식으로 전개되는 어플리케이션들은 새로운 오라클 홈에 포함된 최신 버전이 롤아웃되게 하는 것 이외의 어떤 추가적인 정보 또는 단계를 요하지 않는다. 오라클 홈 디렉토리 외부의 어플리케이션들을 업데이트하는 프로세스는 스테이지식(staged) 및 노-스테이지식(no-staged) 어플리케이션들에 대해 서로 다르지만, 두 경우 모두, 현재 어플리케이션 디렉토리의 위치를 찾고, 해당 디렉토리를 백업 위치로 이동시키고, 새로운 버전의 어플리케이션을 포함하는 어플리케이션 디렉토리를 본래 것의 위치에 이동시키고, 필수적으로는 구 어플리케이션 코드를 새로운 어플리케이션 코드와 교체하는 것을 수반한다. 이러한 동작은 본래의 디렉토리가 액세스되는 동안에는 수행되지 않을 수 있어서, 영향받는 서버들은 이 절차 동안 셧다운되어야 한다. 그러나, 노드 관리자는 어플리케이션 코드에 독립적이기 때문에, 이 프로세스는 (오라클 홈 또는 자바홈을 업데이트하는 것과는 달리) 노드 관리자가 여전히 실행되는 동안 행해질 수 있다. 새로운 오라클 홈을 롤아웃하는 것과 유사하게, 요구되는 어떤 준비(preparation)가 존재한다. 예를 들어, 새로운 어플리케이션 코드를 포함하는 디렉토리는 롤아웃이 시작되기 전에 모든 영향을 받는 노드들에 배포되어야 하고, 이는 각각의 노드에 대해 동일한 경로에 있어야 한다.

[0199] 도 13 내지 16는 하나의 실시예에 따른 패치 이벤트 다이어그램들을 도시한다.

[0200] 스테이지식, 노-스테이지 및 익스터널-스테이지(external-stage) 어플리케이션들이 서로 다르게 전개된다는 사실로 인해, 이들은 적절하게 업데이트되기 위해 서로 다른 처리(treatment)를 요한다. 모든 모드들에서, 새로운 어플리케이션 소스는 어드민 서버 상의 디렉토리로서 제공되어야 한다. 노-스테이지 및 익스터널-스테이지 모드들에서 전개되는 어플리케이션들에 대해, 새로운 어플리케이션 소스는 또한, 어드민 서버 상에 있기 때문에 동일한 경로로 각각의 노드에 사전에 배포되어야 한다.

[0201] 스테이지식 모드(Staged Mode)

[0202] 스테이지식 모드의 실시예에 따르면, 어드민 노드(620) 및 어드민 서버(622) 간의 인터랙션, 및 노드 관리자(624) 및 (MS1 및 MS2로서 표시된) 두 개의 피관리 서버들을 포함하는 노드 1을 도시하는 도 13에 도시된 바와 같이, 스테이지 모드에서 어플리케이션들을 실행시키는 서버들은 어드민 서버로부터 직접적으로 자신의 소스를 얻는다. 어플리케이션들을 업데이트하기 위해, 상기 소스는 어드민 서버 상에서 먼저 업데이트되어야 하고, 그 다음, 서버가 어드민 모드에 있는 동안, 특정 타겟 재전개는, 각각의 서버의 소스를 업데이트하고 변경들을 적절하게 선택(pick up)하도록 각각의 서버를 트리거하기 위해 각각의 서버에 대해 개별적으로 콜될 것이다. 이러한 동작은 일관성을 위해 공통 클러스터 내의 공통 머신 상의 서버들을 함께 그룹화한다.

[0203] 노-스테이지 모드(No-stage Mode)

[0204] 노-스테이지 모드의 실시예에 따르면, 어드민 노드(630)와 어드민 서버(632) 사이의 인터랙션, 및 노드 관리자(634) 및 두 개의 피관리 서버들을 포함하는 노드 1을 마찬가지로 도시하는 도 14에 도시된 바와 같이, 노-스테이지 어플리케이션들은 서버가 시작될 때 서버의 머신 상의 디렉토리로부터 로딩된다. 여기서 어플리케이션 코드를 업데이트하기 위해, 동일한 어플리케이션 디렉토리를 포인팅하는 해당 머신 상의 모든 서버들은 동시에 셧다운되어야 한다. 그 다음, 디렉토리의 컨텐츠는 따로 이동되어 새로운 버전의 어플리케이션과 교체될 수 있다. 디렉토리를 교체함으로써 업데이트가 행해지기 때문에, 시스템은 노-스테이지 어플리케이션들에 대해 공유 스토리지 디렉토리를 이용하는 것을 지원하지 않을 수 있는 바, 이는 이렇게 하는 것이 디렉토리 외부에서 어플리케이션들을 여전히 실행시키는 다른 서버들에 대해 문제를 일으킬 수 있기 때문이다. 그 다음, 영향을 받은 서버들은 어드민 모드에서 시작될 것이며, 특정 타겟 재전개 커맨드가 각각의 서버에 대해 개별적으로 발행되어 각각의 서버가 변경들을 선택하도록 할 것이다.

[0205] 익스터널-스테이지 모드(External-stage Mode)

[0206] 도 15에 도시된 바와 같이, 익스터널-스테이지 모드의 실시예에 따르면, 어드민 노드(640)와 어드민 서버(642) 사이의 인터랙션, 및 노드 관리자(644) 및 두 개의 피관리 서버들을 포함하는 노드 1을 마찬가지로 도시하는 도 15에 도시된 바와 같이, 익스터널-스테이지 어플리케이션들은 이들의 어플리케이션 소스가 워크플로우에 의해

업데이트될 필요가 있다는 것을 고려해 볼 때 노-스테이지 어플리케이션들과 유사하다. 그러나, 주요한 차이점은 익스터널-스테이지 어플리케이션 소스 디렉토리들이 서버의 스테이징 디렉토리에 위치된다는 점이며, 이로 인해, 각각의 서버는 업데이트할 디렉토리의 자신만의 카피를 가진다. 워크플로우는 다른 스테이지 모드들과 마찬가지로 공통 머신 상의 서버들을 함께 셧다운시키고, 그 다음, 각각의 영향받는 서버의 스테이지 디렉토리를 어드민 모드에서 시작하고 그리고 변경들을 선택하도록 서버를 트리거 하기 위해 특정한 타겟 재전개를 이용하기 전에 업데이트할 것이다.

[0207] 상기 프로세스들이 작동하기 위해, 어플리케이션 코드의 교체는 오직 서버들에 대해 행해져야 하는 바, 이들이 셧다운되기 때문이다. 따라서, 동일한 어플리케이션 디렉토리를 공유하는 어떤 서버들은 동시에 셧다운되어야 한다. 이는 서버들이 어플리케이션 디렉토리에 대해 공통의 공유 스토리지 위치를 이용하지 못하게 한다. 각각의 머신은 상기 어플리케이션 디렉토리의 로컬 카피뿐만 아니라 새로운 어플리케이션 디렉토리의 로컬 카피를 가져야 한다. 새로운 어플리케이션 디렉토리, 현재의 어플리케이션 디렉토리 및 백업 위치로의 경로는 모든 영향을 받는 서버들에 대해 동일해야만 한다. 또한, 어플리케이션들은 오라클 홈 디렉토리에 상주하지 못할 수 있다.

[0208] 롤아웃이 진행됨에 따라 그리고 서버들이 요청들을 여전히 서비스하는 동안, 어플리케이션들에 대한 변경들이 스태그드 방식(staggered manner)으로 서버들에 걸쳐 롤아웃 될 것이기 때문에, 롤아웃이 시작되기 전에 생성된 세션들이 어플리케이션의 더 새로운 버전과 호환되지 않을 수 있다. 이는 세션들이 롤아웃 동안 어떻게 처리되는지 그리고 서버들이 어떻게 셧다운되는지에 있어 어떤 복잡성(complication)을 도입하는 바, 이는 어플리케이션을 업데이트하는 것을 지원하는 커맨드들 내의 isSessionCompatible 플래그의 이용을 통해 해소될 수 있다. 구 버전의 어플리케이션들과 새로운 버전의 어플리케이션들 간의 세션들이 호환가능한 경우, 특정한 안전보장조치(safeguards)가 필요치 않을 것이며, 롤아웃은 더욱 효율적으로 완료될 것이다.

[0209] 하나의 실시예에 따르면, 사용자로부터 세 조각의 정보가 일반적으로 요구되는 바, 이는 (config에서 더 많은 정보를 찾기(look up) 위해 이용되는) 어플리케이션 이름, (로컬 디렉토리여야 하는) 새로운/패치된 어플리케이션 코드의 위치, 및 (역시 로컬 디렉토리여야 하는) 현재의 어플리케이션 디렉토리가 백업될 위치이다. 현재의 어플리케이션 소스 위치 및 스테이지 모드는 각각의 서버 및 이의 어플리케이션들의 구성에 기초하여 워크플로우에 의해 계산될 수 있다.

[0210] 심지어, 이러한 감소된 세트의 정보는 커맨드 라인 상에 특정하기가 쉽지 않은 것으로 입증될 수 있다. 이를 해소하기 위해, 하나의 실시예에 따르면, 상기 정보는 커맨드를 발행하기 전에, 상기 커맨드가 실행될 때 상기 커맨드가 판독할 수 있는 위치에 사용자에 의해 텍스트 파일 내로 입력될 수 있다. 각각의 커맨드들에 대한 커맨드-라인 인수는 단순히, 이 파일로의 경로이다.

[0211] 다양한 실시예에 따르면, 파일을 정의하는 서로 다른 포맷들이 이용될 수 있는 바, 주요한 고려사항은 인간이 파일을 생성할 것이기 때문에 파일이 인간 친화적(human-friendly)일 필요가 있다는 것이다. 예를 들어, JSON은 인간 판독가능한 적절한 밸런스를 가지고, 조직(organize)하기 쉽고, 사용자가 각각의 어플리케이션의 속성에 대해 동일한 이름을 사용할 수 있게 하고, 일반적으로 알려져 있고 파싱하기 쉽다는 추가적인 이점을 가진다.

[0212] 롤링 재시작

[0213] 하나의 실시예에 따르면, 제로 다운타임 패치 특징은 한번에 하나씩 서버들의 세트를 재시작하기 위한 메커니즘을 제공한다. 서버들 또는 오라클 홈 또는 도메인 디렉토리 상에서 행해지는 구성 변경이 존재하지 않기 때문에, 서버들은 심지어 공통 오라클 홈 디렉토리로부터 실행되는 동일한 머신 상의 복수의 서버들이 존재하는 경우에도 한번에 하나씩 다운될 것이다. 워크플로우에서 실패(failure)가 존재하는 경우, 이전에 영향을 받은 서버들로 복원하기 위한 본래의 상태가 존재하지 않기 때문에 상기 워크플로우가 복귀될 수 없는 이유도 이 때문이다.

[0214] 진행 모니터링

[0215] 하나의 실시예에 따르면, WLST 롤아웃 커맨드는 롤아웃 태스크의 진행을 모니터링하기 위해 쿼리(query)될 수 있는 WorkflowProgressMBean을 리턴한다.

[0216] 롤아웃 구현

[0217] 하나의 실시예에 따르면, 이 특징은 롤아웃 태스크를 달성하기 위해 여러 고레벨 동작들 또는 패치 프리미티브들을 도입한다. 이 동작들은 워크플로우에서 관리될 수 있도록 오케스트레이션 프레임워크로부터 인터페이스들

을 구현할 수 있다. 패치 프리미티브들은 더 고레벨의 패치 프리미티브들에 의해 콜될 수 있다. 예를 들어, PatchNode 프리미티브는 ShutdownServer, 및 PrepareSwitchOracleHome, RestartNodeManager, AssertSwitchOracleHome, 및 StartServer와 같은 다른 프리미티브들을 콜할 수 있다.

[0218] 하나의 실시예에 따르면, 롤아웃 WLST 콜은, 워크플로우를 생성하여 이를 실행을 위해 워크플로우 수명 관리자(예컨대, WorkflowLifecycleManager)에 패스하기 위해 PatchingFacadeMBean를 이용할 것이다. 상기 워크플로우는 프리미티브들을 통합하는 바, 예컨대, RolloutDirectory, 함께(동일한 클러스터, 동일한 머신) 업그레이드되어야 하는 서버 그룹들을 결정하는 CheckPrerequisites, 그리고 각각의 서버 그룹에 대해서는, 각각의 서버에 대한 (정상적인) ShutdownServer, 노드에 대해 한번 PrepareSwitchOracleHome, 노드에 대해 한번 RestartNodeManager, 노드에 대해 한번 AssertSwitchOracleHome 그리고 각각의 서버에 대해 StartServer을 통합할 것이다.

[0219] 하나의 실시예에 따르면, PatchServer 프리미티브는 한 번에 단일 서버를 패치하는 데 이용가능하다. 그러나, 오라클 홈을 롤아웃하는 것이 디렉토리를 공유하는 노드 상의 서버들 모두에 영향을 끼칠 것이기 때문에, 각각의 영향을 받는 노드 상의 서버들 모두를 포함하는 것이 요구된다. 이는 다른 컴퓨터에 의해 이용되기 위해 또는 부분적인 롤아웃으로부터 회복(recover)하기 위해 제공된다. 이는 영향을 받는 단일 서버들에 대해 프리미티브들을 콜하는 바, 즉 각각의 서버에 대해 (정상적인) ShutdownServer, 노드에 대해 한번 PrepareSwitchOracleHome, 노드에 대해 한번 RestartNodeManager, 노드에 대해 한번 AssertSwitchOracleHome 그리고 각각의 서버에 대해 StartServer을 콜할 것이다.

[0220] 하나의 실시예에 따르면, 오라클 홈 디렉토리가 새로운 이미지에 의해 교체되는 방식은 다음을 포함한다. 1. 서버들이 정상적으로 셧다운된다. 2. PrepareSwitchOracleHome 프리미티브가 콜된다. 이 프리미티브는 해당 노드가 오라클 홈 디렉토리의 스위칭을 행할 스크립트를 셋업하도록 노드 관리자에게 알린다. 이 단계는 노드 관리자가 동작을 수행하기 위해 요하는 모든 파라미터들을 어떻게 얻는지에 관한 것이다. 3. 다음 단계는 RestartNodeManager 프리미티브를 콜하는 것이다. 이는 노드 관리자가 switchOracleHome 스크립트에 대한 제어를 전달할 수 있게 할 것이다. 그 스크립트는 현재의 오라클 홈으로부터 특정된 디렉토리 경로로 이동되고, 본래의 위치 내로 새로운 이미지를 추출하고 그 다음, 노드 관리자를 다시 시작할 것이다. 4. 실행될 다음 프리미티브는 AssertSwitchOracleHome 프리미티브이다. 이 프리미티브는 오라클 홈 디렉토리들의 스위칭이 성공적으로 완료됨을 확인할 것이다. 5. 콜되는 마지막 프리미티브는 StartServers이다. 이 프리미티브는 각각의 서버에 대해 콜되며, ReadyAppCheck가 (구성되는 경우) 성공적으로 리턴될 때까지 완료되지 않을 것이다. 이는, 워크플로우가 어떤 더 많은 서버들을 셧다운하기 전에 모든 어플리케이션들이 요청들을 서비스할 수 있게 할 것이다.

[0221] 에러 및 실패 처리(Error and Failure Handling)

[0222] 오라클 홈 디렉토리들을 업데이트하기 위해 롤링 재시작들을 코디네이션하기 위한 오케스트레이션 프레임워크를 이용하는 장점들 중 하나는, 프로세스가 많은 단계들을 수반할 수 있고 수 시간을 소요할 수 있다는 것이다. 요구되는 단계들을 수동으로 수행하는 것은 지루하고 시간소모적일 수 있고 따라서, 에러들 및 비효율성을 유발할 수 있다. 프로세스를 자동화하는 것은 인간의 에러가 도입될 기회를 감소시키고, 이는 프로세스를 수행하는 데 요구되는 시간이 더욱 효율적으로 이용되게 하고, 이는 여러 가지 실패 처리 옵션들을 제공하고, 그리고 최악의 경우에는, 이는 자체의 변경들 모두를 자체의 본래 상태로 자동으로 복귀시킬 수 있다.

[0223] 하나의 실시예에 따르면, 복수의 커맨드들로 구성된 프리미티브(또는 프리미티브들)을 실행할 때, 실패가 처리될 수 있는 몇 가지 방식들이 존재한다. 개별적인 커맨드의 실패는 무시되거나 또는 프리미티브를 구성하기 위해 이용되는 설정들에 따라 재시도될 수 있다. (파일을 새로운 위치로 이동시키기 전에 본래의 위치로 다시 이동시키는 것과 같은) 로직적 복귀 동작을 갖는 각각의 프리미티브는 또한 커맨드 RevertInterface를 이용하여 복귀 거동을 정의할 수 있다. 복구불가능한 에러(롤아웃 동작의 성공적인 완료를 하지 못하게 하거나 재시도 후 성공하지 못하는 에러)에 직면할 때, 완료된 단계들은 이들이 완료된 역순으로 복귀될 것이다. 이러한 복귀 페이즈 동안 추가적인 실패에 직면하면, 복귀 프로세스는 정지될 것이며, 이 문제는 오퍼레이터에 의해 수동으로 해결될 필요가 있을 것이다.

[0224] 하나의 실시예에 따르면, 사용자는 또한, 실패의 경우에 워크플로우가 자동으로 복귀되지 않아야 함을 특정할 수 있는 바, 이는 사용자에게 상기 워크플로우가 진행되지 못하게 하는 문제를 바로잡을(rectify) 기회를 제공한다. 사용자가 이를 행할 수 있는 경우, 사용자는 정지된 워크플로우 상에서 실행 메소드를 콜할 수 있고, 이는 마지막으로 성공적으로 완료된 커맨드로부터 전방향(forward)으로 이동할 것이다. 사용자가 워크플로우가 실패하도록 하는 에러를 클리어하지 못하는 경우, 사용자는 마지막으로 성공적으로 완료된 커맨드를 시작하여 위

크플로우가 복귀되도록 하기 위해 정지된 워크플로우 상에서 revert를 콜할 수 있다. 워크플로우는 또한 이에 대한 취소(cancel)를 콜하거나 또는 복귀 동안 복구불가능한 에러와 직면함으로써 정지될 수 있다.

[0225] 롤백

[0226] 일부 상황들에서, 이는 오라클 홈의 패치된 버전이 도메인 내의 모든 서버들에 성공적으로 롤아웃되되, 패치된 버전으로 실행되기 전에 패치 자체에 관한 문제가 발견되는 경우일 수 있다. 이 경우, 업데이트를 롤백하고 모든 서버들을 이전 버전으로 이동시키는 것이 바람직할 수 있다. 하나의 실시예에 따르면, 이 동작은 롤아웃 프로세스를 재실행하되, 이전 버전을 타겟 버전으로서 이용함으로써 달성될 수 있다. 어드민 서버가 항상 최상위 패치 레벨에 존재하게 하기 위해, 이는 이전의 패치를 클러스터에 먼저 롤아웃하고 그 다음, 어드민 서버에 개별적으로 롤아웃함으로써 행해져야 한다. 버전을 롤백하는 것에 관한 일부 잠재적인 문제들이 존재하는 바, 예를 들어, 새로운 버전에 도입된 특징들에 대한 구성 정보가 손실될 수 있고, 스키마 변경들을 변경취소(undoing)하는 것은 트랜잭션 데이터가 손실되게 할 수 잇다.

[0227] 패치 Facade들(Patching Facades)

[0228] 하나의 실시예에 따르면, 시스템은 (POJO로서) 패치 facade들 및 PatchingFacadeMBean을 제공할 수 있다. MBean 버전은 비-MBean 버전에 대한 통과 지점(pass-through)으로서 역할을 하지만, pojo 대신 MBean들로서 진행 객체들을 리턴할 것이다. facade 내의 메소드들은, WorkflowLifecycleManager에 패스하기 위해 WorkflowBuilder를 생성하도록 PatchingWorkflowBuilder에서 적절한 메소드들을 콜하는 것을 처리하는 것을 포함하여 오케스트레이션 프레임워크의 지식을 캡슐화한다. 메소드는 노출된 패치 프리미티브들 각각에 대해 제공될 수 있어서 다른 컴포넌트들이 상위 레벨 콜들과 함께 이들을 직접적으로 콜할 수 있게 하고, 상기 상위 레벨 콜들은 여러 프리미티브들을 결합하기 위해 WorkflowBuilders을 생성할 것이다. 메소드들은 또한, 활성 및 완료된 워크플로우들의 리스트를 쿼리(query)할 수 있게 하고 그리고 워크플로우의 이름에 의해 워크플로우에 대한 진행을 찾아보기(look up) 위해 제공될 수 있다. 워크플로우는 시작될 때 콜러에 의해 이름을 할당받는 바, 이 이름은 워크플로우의 진행을 쿼리하기 위해 워크플로우를 식별하는 데 이용될 수 있기 때문에 고유해야만 한다.

[0229] 패치 프리미티브(Patching Primitive)

[0230] 하나의 실시예에 따르면, 패치 프리미티브들은 아웃-오브-플레이스 패치 솔루션에 의해 필요로 되는 롤링 재시작을 정상적으로 수행하기 위해 필요한 동작들이다. 각각의 프리미티브의 리스트, 및 이것이 무엇을 행하는지, 이것이 지원하는 내결함성 메커니즘들 및 이것이 요하는 속성들에 관한 설명이 하기에 기술된다.

[0231] 재시도에 대한 지원 - 프리미티브가 처음에 실패하면 다시 시도되어야 하는 거동을 가지는 경우 이는 참(true)이다. 이는 도래할 서비스와 같이 전환중(transitioning)일 수 있는 다른 객체의 상태에 좌우되는 프리미티브에 대해 또는 신뢰할 수 없는 연결과 같은 간헐적인 실패들을 처리하기 위해 이용될 수 있다.

[0232] 복귀에 대한 지원 - 프리미티브가 속해 있는 워크플로우가 복귀되는 이벤트에서 수행될 수 있는 로직적 "동작취소(undo)" 동작를 가지는 경우 이는 참이다.프리미티브가 복귀 경우에 대해 어떤 특별한 거동을 정의한 경우, 이는 여기 기술될 것이다.

[0233] 커스토마이징된 재개(customized resume) - 워크플로우는 어드민 서버 재시작으로 인해 일시정지된 후 재개될 수 있다. 일부 전제조건들이 여전히 참으로 유지되는지를 확실히 하기 위해 이들을 다시 체크하기 위하여 프리미티브에 표준 재개 기능을 오버라이드할 기회를 허용하는 인터페이스가 존재한다. 프리미티브가 재개의 경우를 위해 어떤 특별한 거동을 정의한 경우, 이는 여기에 기술될 것이다.

[0234] 실패 무시 - 워크플로우의 일부로서 실행되되, 프리미티브가 성공적으로 완료되지 않은 경우 워크플로우가 복귀되지 않게 해야 하는 프리미티브에 대해 이는 참일 것이다. 이는 워크플로우의 성공이 중대하지 않는 동작을 시도하는 프리미티브에 의해 이용될 수 있다.

[0235] 하나의 실시예에 따르면, 각각의 프리미티브는 또한, isDryRun이라 불리는 필드를 체크한다. isDryRun 필드가 참으로 설정된 경우, 프리미티브는 실제로 작업을 수행함이 없이 수행하고자 했었던 작업을 로그할 것이다. 이는 일부 일관성 체크들 역시 수행할 수 있지만, 일부 일관성 체크들은 이 모드에서는 적용되지 않을 수 있다. 예를 들어, StartServer 프리미티브는 StopServer 프리미티브가 실제로 서버를 셧다운시킴을 예상할 수 없고, 따라서 서버가 다운됨을 확인하기 위해 체크를 수행하지 않을 것이다.

[0236] 하나의 실시예에서, 발생할 수 있는 어떤 에러를 진단하고 어떤 노드들 및 서버들에 대해 어떤 프리미티브들이 실행되었는지를 검토함에 있어서 어드미니스트레이터들을 보조하기 위해, 각각의 프리미티브는 어떤 다른 관련

정보와 함께 최상위 레벨 워크플로우의 워크플로우 id, 실행되는 프리미티브의 타입 및 영향받는 타겟들을 나타내는 적어도 하나의 로그 메시지를 서버 로그에 출력하도록 요구된다.

[0237] 예시적인 패치 프리미티브들

[0238] 하나의 실시예에 따르면, 업그레이드들 또는 패치들을 롤아웃하기 위해 이용될 수 있는 예시적인 패치 프리미티브들이 하기에 기술된다. 다른 실시예들에 따르면, 다른 그리고/또는 추가적인 패치 프리미티브들이 지원될 수 있다.

[0239] 셧다운서버

[0240] 하나의 실시예에 따르면, 이 프리미티브는 특정된 피관리 서버(managed server)를 정상적으로 셧다운시킨다. 이는 일반적으로, 피관리 서버가 "실행(RUNNING)" 상태로부터 "셧다운(SHUTDOWN)" 상태로 천이되면서도 프로세스 내의 작업이 정상적으로 처리될 수 있게 하는 장기 실행 프로세스이다. 프리미티브는 기본적으로, WLS 내의 정상적인 셧다운 특징에 의존한다. 서버를 실제로 셧다운시키기 전에, 프리미티브는 서버의 현재 상태(서버가 실행, 셧다운, 어드민 또는 스탠바이 상태인지)를 획득하고, lastServerState로 불리는 공유된 상태 속성을 업데이트할 것이다. 이는 적어도 서버가 시작되어야 하는지를 결정하기 위해 StartServer 프리미티브에 의해 이용될 것이다. ShutdownServer 프리미티브가 실행된 때 서버가 정지되지 않은 경우, StartServer 프리미티브는 서버를 시작하지 않을 것이다.

[0241] 파라미터들

[0242] 프리미티브들에 대한 파라미터들은 어떤 공유된 상태 객체들과 같이 이름에 의해 패스된다. 이름에 대한 파라미터들 및 공유된 상태 객체들의 테이블이 아래에 제시된다.

표 6

[0243]

serverName	셧다운될 필요가 있는 서버들의 이름
ignoreSessions	세션들이 완료되거나 타임아웃되기를 기다리기 보다는 이 세션들을 즉시 드롭시킴. 이 파라미터는 서버가 이미 중지(quiesce)된 경우 특정될 필요가 없다.
shutdownTimeout	서버가 정상적인 셧다운을 완료하기 위한 (초단위의) 시간 제한. 디폴트는 타임아웃이 없음을 나타내는 0.
lastServerState	서버가 셧다운되기 전에 StartServer 프리미티브에 의해 이용될 서버의 상태를 저장. 가능한 값들은 실행(RUNNING), 셧다운(SHUTDOWN), 어드민(ADMIN) 또는 스탠바이(STANDBY).
isDryRun	작업이 행해지지 않아야 하되, 프리미티브가 행하려 했었던 것을 로그해야 하는 경우 참.

[0244] 내결함성 지원(Fault Tolerance support)

표 7

[0245]

Support for retry	재시도를 지원
Support for revert	복귀를 지원. 복귀 동작은 StartServer 프리미티브/커맨드를 인보크할 것이다.
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0246] UpdateOracleHomeDirectory

[0247] 하나의 실시예에 따르면, 이 프리미티브는 오라클 홈 디렉토리를 새로운 디렉토리의 컨테츠들로 업데이트하는 작업을 수행한다. 현재의 오라클 홈 위치로부터 실행중인 어떤 프로세스가 먼저 셧다운되어야 한다. 노드 관리자는 외부 스크립트를 수동 제어(hand control)하는 바, 이는 일단 준비가 되면(in place), 새로운 디렉토리로부터 이를 재시작할 것이다.

[0248] 파라미터들

[0249] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 8

[0250]

newDirectory	롤아웃할 새로운 이미지. 이는 확장된 디렉토리이거나 또는 FMW 이동 스크립트 copyBinary로 생성된 아카이브된 디렉토리일 수 있다.
backupDirectory	현재의 오라클 홈 디렉토리가 향후 복귀 또는 롤백을 위해 필요로되는 경우 재할당될 경로
machineName	디렉토리가 업데이트될 머신의 이름
timeoutMilliseconds	타임아웃하거나 에러를 보고하기 전에 UpdateOracleHomeLink 스크립트가 실행될 수 있게 하는 시간의 길이
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0251] 내결함성 지원

표 9

[0252]

Support for retry	가능한 재시도, 단순히 스크립트를 다시 콜
Support for revert	복귀가 오라클 홈을 본래대로 변경
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0253] PrepareSwitchOracleHome

[0254] 하나의 실시예에 따르면, 이 프리미티브는 노드 관리자에게 오라클 홈 디렉토리를 대체하고 노드 관리자를 재시작하는 데 사용될 스크립트를 설정하기 위해 필요한 파라미터들을 제공한다

[0255] 파라미터들

[0256] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 10

[0257]

MachineName	노드 관리자가 실행되는 머신빈의 이름
newDirectory	롤아웃할 새로운 이미지. 이는 확장된 디렉토리이거나 또는 FMW 이동 스크립트 copyBinary로 생성된 아카이브된 디렉토리일 수 있다.
backupDirectory	현재의 오라클 홈 디렉토리가 향후 복귀 또는 롤백을 위해 필요로되는 경우 재할당될 경로
timeoutMillis	노드 관리자를 재시작 한 후 클라이언트가 노드 관리자에 재연결하기 위해 대기해야 하는 시간. 초과되는 타임아웃은 실패된 태스크 및 도달가능하지 않은 노드 관리자를 고려할 것이다. 디폴트는 3분이다.
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0258] 내결함성 지원

표 11

[0259]

Support for retry	없음(none)
Support for revert	복귀 거동은 AssertSwitchOracleHome의 실행 거동과 동일함
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0260] AssertSwitchOracleHome

[0261] 하나의 실시예에 따르면, 프리미티브는 노드 관리자가 재시작 한 후 사용되어 오라클 홈이 성공적으로 업데이트되었는지를 확인한다. 이는 업데이트가 성공적이면 참을 리턴하고, 그렇지 않으면 실패한다.

[0262] 파라미터들

[0263] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 12

[0264]

MachineName	노드 관리자가 실행되는 머신빈의 이름
newDirectory	롤아웃할 새로운 이미지. 이는 확장된 디렉토리이거나 또는 FMW 이동 스크립트 copyBinary로 생성된 아카이브된 디렉토리일 수 있다.
backupDirectory	현재의 오라클 홈 디렉토리가 향후 복귀 또는 롤백을 위해 필요로되는 경우 재할당될 경로
timeoutMilliseconds	노드 관리자를 재시작 한 후 클라이언트가 노드 관리자에 재연결하기 위해 대기해야 하는 시간. 초과되는 타임아웃은 실패된 태스크 및 도달가능하지 않은 노드 관리자를 고려할 것이다. 디폴트는 3분이다.
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0265] 내결함성 지원

표 13

[0266]

Support for retry	없음
Support for revert	복귀 거동은 PrepareSwitchOracleHome의 실행 동작과 동일함
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0267] StartServer

[0268] 하나의 실시예에 따르면, 이 프리미티브는 (새로운 경로 위치를 사용하여) 관리 서버를 시작한다. 서버는 여기에 문서화된 바와 같이, 스탠바이(STANDBY), 어드민(ADMIN) 또는 실행(RUNNING) 모드에서 시작하도록 구성될 수 있다. 이 정보는 구성에 유지되며 다음에 서버가 (재)시작 될 때 사용된다. 이 프리미티브를 통해 서버가 시작될 때, 이는 시작되도록 구성된 모드에 자동적으로 천이될 것이다. 디폴트 서버 시동 상태는 실행 상태이다.

[0269] 하나의 실시예에 따르면, 이 프리미티브는 또한, ShutdownServer 프리미티브가 콜되었을 때 서버가 이미 셧다운(SHUTDOWN) 상태에 있는지를 알기 위해 lastServerState 공유 속성의 값을 체크한다. 만일 그러하다면, 원래 상태를 보존하길 원하기 때문에 StartServer 프리미티브는 서버를 시작하지 않을 것이다.

[0270] 파라미터들

[0271] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 14

[0272]

serverName	시작될 피관리 서버의 이름
lastServerState	서버가 셧다운되기 전에 서버의 상태를 포함. ShutdownServer 프리미티브가 실행되기 전에 서버가 SHUTDOWN이면, ShutdownServer 프리미티브는 이를 시작하지 않을 것이다. 가능한 값들은 RUNNING, SHUTDOWN, ADMIN, 또는 STANDBY이다.
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0273] 내결함성 지원

표 15

[0274]

Support for retry	이 프리미티브는 재시도를 지원하지 않는다. 이는 서버를 재시작하기 위해 노드 관리자를 이용하며, 노드 관리자에는 이미 재시도 로직이 구축되어 있다.
Support for revert	ShutdownServer 프리미티브를 인보크할 복귀를 지원

Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0275] RestartNodeManager

[0276] 하나의 실시예에 따르면, 이 프리미티브는 노드 관리자를 재시작 할 것이다. Java 기반의 노드 관리자 프로세스는 startNodeManager 스크립트에 의해 인식되는 특정한 리턴 코드로 종료(exit)될 것이다. 이 반환 코드를 참조(see)할 시, startNodeManager 스크립트는 updateOracle Home 스크립트를 킥오프(kick off)할 것이다. 이 스크립트는 도메인 디렉토리에 상주하며, 현재의 오라클 홈 디렉토리를 특정된 백업 위치로 이동시키고 새로운 오라클 홈 디렉토리를 (새 디렉토리가 디렉토리가 아닌 아카이브인 경우 pasteBinary 사용하여) 장소로 이동시키는 역할을 한다. 그 다음, 이는 새로운 오라클 홈 디렉토리로부터 노드 관리자를 시작한다. updateOracleHome 스크립트가 아카이브를 추출하거나 새 디렉토리를 장소로 이동하는 에러에 직면하면, 이는 본래의 디렉토리를 장소로 이동하고 노드 관리자를 시작할 것이다.

[0277] 파라미터들

[0278] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 16

[0279]

MachineName	노드 관리자가 실행되는 머신빈의 이름
NMRestartTimeoutMilliseconds	옵션적임. 노드 관리자를 재시작 한 후 클라이언트가 노드 관리자에 재연결하기 위해 대기해야 하는 시간. 초과되는 타임아웃은 실패된 태스크 및 도달가능하지 않은 노드 관리자를 고려할 것이다. 디폴트는 3분이다.
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0280] 내결함성 지원

표 17

[0281]

Support for retry	노드 관리자가 여전히 도달가능하면, 재시도를 할 수 있다. 그렇지 않으면, 재시도 옵션이 없다.
Support for revert	복귀는 노드 관리자를 다시 재시작할 것이다.
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0282] ExecScript

[0283] 하나의 실시예에 따르면, 이 프리미티브는 특정된 머신 상의 도메인/빈/패치 디렉토리로부터 커스텀 스크립트를 실행시킨다.

[0284] 파라미터들

[0285] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 18

[0286]

MachineName	노드 관리자가 실행되는 머신빈의 이름
ScriptName	실행할 스크립트의 이름
ScriptEnv	옵션적인 스크립트 env로서 패스할 수 있는 값들의 일부 맵
ScriptOutputLocation	스크립트 출력, 즉 로깅 또는 파일 또는 다른 것을 기록할 위치
ScriptExecutionTimeout	스크립트 exec이 완료되기 위해 대기하는 밀리초. 일단 특정된 시간이 경과하면, 스크립트 프로세스는 중단(halt)되고 노드 관리자는 타임아웃을 나타내는 에러를 리턴한다. 디폴트는 완료까지 블록킹하는 0이다.

isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0287] 내결함성 지원

표 19

[0288]

Support for retry	재시도할 수 있다
Support for revert	복귀는 복귀가 그 특정 워크플로우/프리미티브로 어떻게 작업할 수 있는지를 특정하기 위해 복귀 메소드를 오버라이드하는 누군가에 의해 지원될 수 있다.
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0289] UpdateNodeDirectory

[0290] 하나의 실시예에 따르면, 이 프리미티브는 개별 노드에 대한 오라클 홈 디렉토리를 업데이트하는 데 요구되는 모든 프리미티브들을 콜한다. 이는 ShutdownServer, UpdateOracleHomeDirectory, PrepareSwitchOracleHome, AssertSwitchOracleHome, RestartNodeManager, StartServer를 콜할 것이다.

[0291] 파라미터들

[0292] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 20

[0293]

machineName	업데이트할 노드의 이름
rolloutMode	롤아웃의 모드, 도메인(DOMAIN), 클러스터(CLUSTER) 또는 서버(SERVER)
domainName	영향을 줄 도메인의 이름
clusterName	영향을 줄 클러스터의 이름
serverNames	업데이트할 서버들의 이름들
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0294] 내결함성 지원

표 21

[0295]

Support for retry	이 프리미티브 자체는 재시도를 지원하지 않지만, 이 프리미티브가 콜하는 프리미티브들 중 일부는 재시도를 지원할 수 있다.
Support for revert	이 프리미티브에 대한 복귀 메소드는 이 프리미티브의 일부로서 실행된 프리미티브들의 전부에 대해 복귀를 콜할 것이다.
Customized resume	커스텀 거동 안함(no custom behaviour)
Ignore failures	아니오(no)

[0296] RolloutDirectory

[0297] 하나의 실시예에 따르면, 이는 도메인 또는 클러스터에 걸쳐 오라클 홈 업데이트를 롤아웃하기 위한 주요 최상위 레벨 프리미티브이다. 이는 롤아웃이 성공적이게 하기 위해 다른 모든 프리미티브를 조정한다. 이는 롤아웃 모드를 고려하여 업데이트 할 서버들을 결정하고, 서버들 및 노드 관리자가 올바른 시퀀스로 업데이트되게 한다. 이는 성공을 하지 못하게 할 수 있는 어떤 구성 문제들을 신속하게 발견하기 위한 시도의 제1 단계로서 checkPrerequisites를 콜할 것이다. 그 다음, 이는 각 노드에 대해 정확한 순서로 UpdateNode를 콜할 것이다

[0298] 파라미터들

[0299] 프리미티브에 대한 파라미터들은 어떤 공유된 상태 객체들과 마찬가지로 이름에 의해 패스된다. 이름에 의한 파라미터들 및 공유된 상태 객체들의 테이블이 하기에 제시된다.

표 22

[0300]

newDirectory	롤아웃할 새로운 이미지. 이는 확장된 디렉토리이거나 또는 FMW 이동 스크립트 copyBinary로 생성된 아카이브된 디렉토리일 수 있다.
backupDirectory	현재의 오라클 홈 디렉토리가 향후 복귀 또는 롤백을 위해 필요로되는 경우 재할당될 경로
rolloutMode	롤아웃의 모드, 도메인(DOMAIN), 클러스터(CLUSTER) 또는 서버(SERVER)
machineName	업데이트할 노드의 이름
domainName	영향을 줄 도메인의 이름
clusterName	영향을 줄 클러스터의 이름
serverNames	업데이트할 서버들의 이름들
isDryRun	어떤 작업도 행해지지 않아야 하되, 프리미티브가 행하고자 했던 것을 로그해야만 하는 경우 참

[0301] 내결함성 지원

표 23

[0302]

Support for retry	이 프리미티브 자체는 재시도를 지원하지 않지만, 이 프리미티브가 콜하는 프리미티브들 중 일부는 재시도를 지원할 수 있다.
Support for revert	이 프리미티브에 대한 복귀 메소드는 이 프리미티브의 일부로서 실행된 프리미티브들의 전부에 대해 복귀를 콜할 것이다.
Customized resume	재개 기능은 디폴트일 것이다.
Ignore failures	아니오(no)

[0303] 심볼적 링크들(Symbolic Links)

[0304] 전형적인 시스템에서, 도메인에 오라클 홈의 위치가 참조되는 어떤 위치들이 존재할 수 있다. 이는 시작 스크립트들, 속성 파일들 및 xml 구성 파일들 내의 변수를 포함한다. 하나의 실시예에 따르면, 오라클 홈 디렉토리에 대한 경로에서 심볼적 링크를 사용하는 것은 시스템이 단순히 심볼적 링크를 변경함으로써 오라클 홈의 위치를 업데이트 할 수 있게 한다. 이 방식으로, 시스템이 경로가 변경될 때 상기 경로를 참조하는 모든 파일을 추적하고 업데이트할 필요가 없다. 각 노드에서 오라클 홈을 포함하는 공유 저장소는 잠재적으로 공유 저장 디바이스 상의 공통 디렉토리에 인스톨된 복수의 어플리케이션 서버, 예컨대 WLS, 버전들을 노출하는 레벨에 마운트된다. 이 방식으로, 새로운 오라클홈 디렉토리들이 생성 및 패치될 수 있으며 노드들 중 어느 것 상의 마운트 포인트들을 변경해야 함이 없이 이용가능해질 것이다. symlink는 마운트 디렉토리를 통해 어플리케이션 서버의 특정 버전을 포인팅하도록 생성된다.

[0305] 공유 저장소 상의 홈

[0306] 하나의 실시예에 따르면, 롤아웃 오케스트레이션 태스크를 실행하기 위한 전구체(precursor)로서 복제 및 패치되어야 하는 디렉토리의 수를 최소화하기 위해, 오라클 홈이 패치될 모든 서버들에 의해 액세스가능한 공유 저장 디바이스 상에 위치되도록 권고된다. 이 방식으로, 단일 복제본이 만들어지고 패치될 수 있으며, 모든 서버들이 동일한 저장 포인트를 마운트할 수 있다. 제공된 저장소에는 일부 리던던시가 구성되어서, 모든 서버에 대해 단일 실패 포인트가 되지 않도록 권고된다. 또한, 모든 서버들이 동일한 경로를 사용하여 공유 저장소 이미지를 마운트해서, 각 서버에 대한 심볼적 링크가 동일한 방식으로 업데이트될 수 있도록 하는 것이 요구된다.

[0307] 별개의 시스템 상의 클러스터 내의 서버

[0308] 상기 기술된 바와 같이, 하나의 실시예에 따르면, 서버를 업그레이드하는 동안 가동 시간(uptime)을 유지하기 위한 인자(factor)는 클러스터들로 구성된 고 가용성의 장점을 취한다. 하나의 실시예에 따르면, 클러스터 내의 최소 수의 서버들은 항상 동작적으로 유지되어야 한다. 동일한 머신 상의 클러스터 내의 서버들이 (공통 도메인 디렉토리 및 symlink를 공유하는 경우) 함께 재시작될 필요가 있기 때문에, 클러스터 내의 서버들은 적어도 2개의 다른 물리적 머신들 상에 호스트되어야 하되, 클러스터 당 최소 3 개의 머신들이 권고된다. 이는 일부가 업 상태(up)를 유지하여 서비스를 제공할 수 있게 하며, 나머지는 롤링 재시작의 일부로서 다운된다. 요청들을 처리하기 위한 다른 머신들 상에서 이용가능한 서버들의 수를 결정할 때, 실행 중이되 어드민 또는 스탠바이 모드에 있는 피관리 서버는 요청들에 응답하지 않을 것이기 때문에 이들을 배제하는 것이 중요하다.

[0309] 어드민 서버 분리

[0310] 어드민 서버 및 피관리 서버가 동시에 업데이트될 필요가 있는 경우, 롤아웃 프로세스는 매우 복잡할 수 있다. 예를 들어, 어드민 서버와 피관리 서버가 동일한 머신 상에서 실행되고 동일한 도메인 디렉토리를 공유하도록 구성된 경우가 그러한 경우일 수 있다. 어드민 서버는 공유된 심볼적 링크로부터 실행 중일 수 있기 때문에 피관리 서버와 동시에 다운되어야 할 수 있다. 이러한 제약은 피관리 서버 단위로 패치들을 롤아웃 할 수 있도록 피관리 서버의 인스톨레이션 홈들을 격리시킴으로써 해결될 수 있다. 하나의 실시예에 따르면, 이 문제를 단순화시키는 2 개의 다른 구성들이 지원된다.

[0311] 제1 구성은 어떤 피관리 서버가 머신 상에서 실행됨이 없이 상기 머신 상에서 어드민 서버가 실행되게 하는 것이다. 이는, 어드민 서버가 스스로 한 단계 업데이트될 수 있게 하고, 일단 완료되면, 다음 단계는 다른 머신들 상에서 해당 도메인의 피관리 서버를 업데이트하는 것일 것이다.

[0312] 제2 구성은 어드민 서버가 피관리 서버와 동일한 머신 상에서 실행될 수 있게 하되, 어드민 서버가 자체 별개의 도메인 디렉토리 밖에서 실행되도록 하는 것이다. 이는 다시, 어드민 서버가 개별적으로 업데이트될 수 있게 하고, 피관리 서버는 자체 단계에서 업데이트될 수 있다.

[0313] 클러스터-레벨 패치

[0314] 하나의 실시예에 따르면, 도메인 내의 모든 서버들을 업데이트할 메커니즘을 제공하는 것에 추가적으로, 시스템은 도메인 내의 개별 클러스터를 업데이트기 위한 능력을 제공할 수 있다. 사용자가 클러스터 롤아웃 모드를 사용하려고 시도할 때, 여러 클러스터들을 서빙하는 단일 서버 상에 복수의 피관리 서버들이 존재하는 경우, 피관리 서버들은 자신이 서빙하는 클러스터에 따라 별개의 도메인 디렉토리들을 가져야 한다. 이는, 클러스터에 대한 노드 상의 모든 피관리 서버들이 다운되고 다른 클러스터를 서빙하는(그리고 여전히 실행되는) 피관리 서버들의 symlink에 영향을 끼침이 없이 symlink로 하여금 업데이트되게 하도록 요구된다.

[0315] 롤아웃 모드들

[0316] 하나의 실시예에 따르면, 롤아웃은 서버를 정상적으로 셧다운하고 이의 오라클 홈 symlink를 변경하고, 그리고 이를 다시 시작하는 것을 포함한다. 이는 전체 도메인, 도메인 내의 단일 클러스터, 또는 개별 서버들에 적용될 수 있다. 이러한 모드들 중 어느 것에 대해, 공통 오라클 홈을 공유하는 단일 머신 상에서 업데이트되는 복수의 서버들이 존재하는 경우, 이들은 함께 셧다운되고 업데이트될 것이다. 또한, 서버의 오라클 홈이 업데이트될 때, 이의 관련 노드 관리자가 변경들을 적용(pick up)하기 위해 재시작될 것이다. 이러한 것이 엄격하게 필수적인 것은 아닌 경우들이 존재할 수 있지만, 이를 일관되게 행하는 것은 프로세스를 간략화시키고 결과적으로 오직 노드 관리자가 응답하지 않는 시간의 윈도우가 짧아지게 한다.

[0317] 하나의 실시예에 따르면, 도메인 모드 롤아웃은 도메인 내의 어드민 서머 및 모든 피관리 서버들을 이들의 관련된 모든 노드 관리자와 함께 업데이트 할 것이다. 어드민 서버가 항상 이의 피관리 서버 중 어느 것의 최상위 패치 레벨에서 실행되는 것이 중요하다. 도메인 모드 롤아웃 중에 이 요건이 충족되게 하기 위해, 어드민 서버는 항상 피관리 서버들 전에 업데이트될 것이다.

[0318] 하나의 실시예에 따르면, 클러스터 모드 롤아웃은 어드민 서버를 업데이트하지 않고, 클러스터 내의 모든 피관리 서버들 및 그들의 관련 노드 관리자들을 업데이트 할 것이다.

[0319] 하나의 실시예에 따르면, 서버 모드 롤아웃은 타겟 파라미터에 특정된 서버들에 영향을 미칠 것이다. 또한, 이는 이 서버들과 관련된 노드 관리자들을 업데이트할 것이다.

[0320] WLST 커맨드 롤아웃

[0321] 하나의 실시예에 따르면, 롤아웃 작업은 어떤 서버들이 어떤 순서로 업데이트 될 필요가 있는지를 파악하고, 이 서버들을 안전하게 업데이트 할 워크플로우를 생성하는 것을 담당한다. 이는 노드를 중지(quiesc)시키는 것, 서버를 정상적으로 셧다운시키는 것, 오라클 홈 링크를 업데이트시키는 것, 노드 관리자를 재시작하는 것, 서버를 시작하는 것 및 노드를 정상적으로 활성화시키는 것을 포함한다. 롤아웃 태스크는 결과 MBean이 나중에 액세스될 수 있거나 다른 WLST 연결에 의해 액세스될 수 있도록 워크플로우 수명 주기 관리자 (예컨대, WorkflowLifeCycleManager, LCM)에 등록할 이름을 취한다. 롤아웃 태스크는 상태에 대해 폴링될 수 있는 WorkflowProgressMBean을 리턴할 것이다. 일부 예들이 아래에 제공된다.

[0322] 도메인에 걸쳐 롤아웃을 수행:

[0323] > progress = rollout('Domain1Rollout', /opt/OracleHome, /mnt/wls1214.01)

[0324] 클러스터에 걸쳐 롤아웃을 수행:

[0325] > progress = rollout('Cluster1Rollout', /opt/OracleHome, /mnt/wls1214.01, 'Cluster', 'Cluster1')

[0326] 두 개의 특정 서버들에 롤아웃을 수행:

[0327] > progress = rollout('MSRollout', /opt/OracleHome, /mnt/wls1214.01, 'Server', 'managedServer1, managedServer2')

[0328] 구성된 OTD 없이 도메인에 걸쳐 드라이 런(dry run) 또는 롤아웃을 수행:

[0329] > progress = rollout('Domain1Rollout', /opt/OracleHome, /mnt/wls1214.01, 'Domain', 'Domain1','isDryRun=true, useOTD=false')

[0330] 하나의 실시예에 따르면, WLST 롤아웃 커맨드는 롤아웃 태스크의 진행을 모니터하기 위해 쿼리될 수 있는 WorkflowProgressMBean을 반환한다. 이 정보는, 재연결할 필요가 있고 또한 워크플로우가 완료된 후에도 이용가능하게 유지되는 WLST 세션에게 이용가능하다.

[0331] 노드 관리자

[0332] 하나의 실시예에 따르면, 자동 패치 롤아웃 솔루션은 원격 머신들 상의 환경들을 업데이트하기 위한 메커니즘을 필요로 한다. 하나의 실시예에 따르면, 오케스트레이션 프레임 워크는 어드민 서버로부터 실행될 수 있고 각 머신 상의 노드 관리자에게 위임하여 오라클 홈을 업데이트하고 새로운 바이너리를 활용(uptake)하기 위해 프로세스들을 재시작하는 것과 같은 태스크들을 이행할 수 있다.

[0333] 하나의 실시예에 따르면, 노드 관리자는 원격 머신 상에서 커스텀 패치 스크립트를 실행하여 오라클 홈으로의 심볼적 링크를 변경하기 위한 메카니즘으로서 기능 할 것이다. 스크립트는 도메인 당 머신 당 한 번 실행될 수 있다. 노드 관리자는 자동화된 서비스 마이그레이션 동안 기본 스크립트 실행을 할 수 있게 하기 위해 내부적으로 사용되는 API를 지원하는 바, 이는 상기 기술된 패치 기능을 지원하기 위해 레버리지 될 수 있다.

[0334] 하나의 실시예에 따르면, 심볼적 링크는 노드 관리자가 실행되는 동안 스위칭 될 것이지만, startNodeManager 스크립트들은 항상 심볼적 링크를 사용기 보다는, 실제 디렉토리 밖에서 실행되도록 설정될 것이다. 심볼적 링크는 패치된 바이너리를 활용할 수 있도록 노드 관리자를 재시작하는 데만 사용될 것이다. 도메인 또는 오라클 홈 외부의 노드 관리자 홈에 있는 페어런트 시작 스크립트는 심볼적 링크 위치를 사용하여 기본(base) startNodeManager 스크립트를 실행할 것이다. 상기 기본 스크립트는 실제 디렉토리에 설정된 WL_HOME과 함께 인스톨되며 모든 환경 값들은 해당 값을 사용하여 생성된다. 결과적으로, 도메인은 심볼적 링크 위치에서 실행되며, 노드 관리자는 오직 실제 디렉토리로부터 실행될 것이고 따라서, 심볼적 링크가 전환될 때 영향을 받지 않을 것이다.

[0335] 하나의 실시예에 따르면, 노드 관리자로부터 실행되는 시스템 컴포넌트들은 그들의 프로세스들이 패치를 지원할 수 있게 하기 위한 옵션들을 가질 것이다.

[0336] 먼저, 이들이 노드 관리자 환경을 사용하여 이들의 프로세스들을 시작하면, 이들은 심볼적 링크 변경으로부터 격리되고 노드 관리자 버전과 일관될 것이다. 이는, 심볼적 링크가 변경되는 동안 이들의 컴포넌트가 계속 실행될 수 있게 하고 새로운 오라클 홈 위치를 적용(pick up)하기 위해 오직 노드 관리자가 재시작된 후에 재시작될 수 있음을 의미한다.

[0337] 둘째로, 이들이 심볼적 링크를 보다 직접적으로 사용하고자 한다면, 이들은 WLS 사용과 같은 일부 시작 스크립트를 통해 도메인 자체로부터의 값 또는 LINK_MW_HOME과 같은 정의된 값으로서 노드 관리자 환경으로부터의 값을 얻을 필요가 있을 것이며, 이들의 프로세스가 심볼적 링크가 변경 전에 적절하게 셧다운되도록 할 필요가 있을 것이다. 또 다른 옵션은 이들이 자신의 경로 정보를 공급하고 이를 직접 관리할 수 있도록 하는 것이다. 예를 들어, OHS 인스톨은 "ohs.home"을 JAVA_OPTIONS 환경 플래그의 노드 관리자에게 패스한다. 이 값은 경로가 변경될 때 그리고 프로세스가 재시작될 때를 제어하는 자체 패치 프리미티브를 제공함으로써 패치 동안 관리되는 심볼적 링크일 수 있다.

[0338] 하나의 실시예에 따르면, 자동 롤아웃 패치의 일부로서, 노드 관리자는 예를 들어 "RESTART" 커맨드를 노드 관리자에게 발행함으로써 새로운 (패치된) WebLogic Server 이미지로부터 실행(runs off)될 수 있도록 재시작될

수 있다. 노드 관리자는 또한, 다른 옵션들을 특정하는 사용자 공급 스크립트와 같은 다른 방식들로 시작될 수 있다. 하나의 접근법은 종료 코드를 캡쳐하고 그 다음 심볼적 링크 위치에서 발견된 startNodeManager 스크립트를 실행하기 위해 기본 exitNodeManager 스크립트에 의존하는 것이다. 예를 들어, 유입 RESTART 커맨드는 코드 88로 JVM을 종료한다. 스크립트는 88을 보고, 스크립트 자체에 대한 어떤 변경들을 적용하기 위해 새로운 스크립트를 사용하여 다른 인스턴스를 시작하는 것을 시도한다. 이는, WL_HOME/server/bin 하의 기본 startNodeManager 스크립트에 대해서만 도메인 레벨 또는 다른 래퍼 스크립트들에 어떤 변경들을 적용하지 않을 것이다. 이는, 이 특정 토폴로지에서 심볼적 링크가 되는, 페어런트 스크립트에 의해 사용된 SCRIPTPATH를 실행함으로써 달성된다.

[0339] 하나의 실시예에 따르면, 자동 패치 롤아웃 솔루션에서, 롤아웃 커맨드는 모든 피관리 서버를 셧다운하고, 노드 관리자를 통해 커스텀 패치 스크립트를 실행하고, 모든 피관리 서버들을 시작하고, 노드 관리자를 재시작할 것이다. 노드 관리자 자체는, System.getenv () API 및/또는 ProcessBuilder.environment () API를 통해 시스템 속성들을 획득하고 이 값들을 새로운 프로세스가 생성될 때 이 새로운 프로세스에 구성된 값들과 함께 제공함으로써 자체 환경을 패스한다.

[0340] 하나의 실시예에 따르면, 도메인은 노드 관리자가 자신의 오라클 홈 디렉토리의 원래 뷰를 유지하는 동안 스왑될 수 있는 오라클 홈 디렉토리에 대한 자신만의 고유의 심볼적 링크를 가진다. 이러한 토폴로지에서 노드 관리자는 부정확한 버전으로부터 바이너리들에 피관리 서버 포인터들을 제공할 수 있는 CLASSPATH 및 다른 값을 패스한다. 이는 WebLogic Server 및 오라클 홈에 특정적이지 않은 환경 값을 패스함으로써 해결될 수 있다.

[0341] 하나의 실시예에 따르면, 도메인 당 노드 관리자 및 머신 당 노드 관리자 모두에서, NodeManagerHome 디렉토리는 오라클 홈 디렉토리 외부에 위치될 것으로 예상된다. 디폴트로, 도메인 당 노드 관리자의 NodeManagerHome 디렉토리는 도메인 디렉토리 하의 서브디렉토리이다.

[0342] 노드관리자 재시작

[0343] 하나의 실시예에 따르면, 시스템은 자바 기반의 노드 관리자 프로세스를 재시작하기 위한 자동화된 능력(capability)을 제공할 수 있다.

[0344] 자바 기반의 노드 관리자

[0345] 하나의 실시예에 따르면, 자바 기반의 노드 관리자는 NMClient로부터 발행되는 새로운 커맨드 "RESTART"을 수락할 것이다. NMServer는 restart 커맨드를 수신할 때, 특정 종료 코드 88로 종료될 것이다. 어떤 정상적인 셧다운 동작이 역시 취해져야 하되, 노드 관리자에 의해 시작된 피관리 프로세스는 실행 상태를 유지해야 한다. NMClient API는 다음을 제안한다.

[0346] /**

[0347] * Issue the RESTART command to the NMServer

[0348] * @param timeoutMillis the amount of time to wait for theNodeManager

[0349] * process to be restarted and reachable before throwing anIO Exception

[0350] * a value of 0 will return without blocking. Values must bepositive.

[0351] */

[0352] public void restart(long timeoutMillis) throws IO Exception;

[0353] startNodeManager 스크립트

[0354] 하나의 실시예에 따르면, 자바 노드 관리자가 더이상 실행되지 않을 때, 공급된 startNodeManager 스크립트는 특정 코드, 88을 체크할 것이다. 88이 리턴된 코드일 때, 스크립트는 심볼적 링크 위치에서 발견되는 새로운 startNodeManager 스크립트를 런칭할 것이다. 바이너리들 및 스크립트들을 포함하는 모든 새로운 패치 파일은 별개의 위치에 위치될 것이며 심볼적 링크를 사용하여 이용가능할 것이다. 이는 파일들 중 어느 것도 겹쳐 쓰여지지(overwritten) 않아야 함을 의미한다. 재시작 시나리오는 다음 예제와 같이 스크립트될 수 있는 바, $WL_HOME은 심볼적 링크 위치를 포인팅한다.

[0355] "${JAVA_HOME}/bin/java" ${JAVA_PROPERTIES}

[0356] weblogic.NodeManager

[0357] if [$? -eq 88]; then

[0358] exec ${SCRIPT_PATH}/startNodeManager.sh

[0359] fi

[0360] 하나의 실시예에 따르면, 노드 관리자 프로세스를 시작하는 많은 다른 방법들은 WL_HOME/server/bin 디렉토리에 포함된 기본 startNodeManager 스크립트를 사용할 수 있다. domain/bin의 도메인 레벨 스크립트 및 커스텀 래퍼는 이 스크립트에 위임 해야하고 그 결과, 런칭을 위해 동일한 로직을 사용해야 하며, WLST startNodeManager 커맨드 역시 이 스크립트들을 사용할 수 있다.

[0361] 도 16은 하나의 실시예에 따른, 패치를 위한 방법의 흐름도를 도시한다.

[0362] 도 16에 도시된 바와 같이, 단계(660)에서, 하나 이상의 파티션들을 지원하는 소프트웨어 어플리케이션의 실행을 위한 도메인을 포함하는 하나 이상의 컴퓨터에서 어플리케이션 서버 환경이 제공되며, 각 파티션은 도메인의 어드미니스트레이티브 및 런타임 서브디비전(dministrative and runtime subdivision)을 제공하고, 파티션은 옵션에 따라서는, 전개가능한 어플리케이션들 또는 리소스들의 집합을 가지고 그리고/또는 리소스 그룹 템플릿을 참조하는 하나 이상의 리소스 그룹들을 포함할 수 있다.

[0363] 단계(662)에서, 어플리케이션 서버, 어플리케이션, 또는 다른 컴포넌트가 실행되는 하나 이상의 컴퓨터 노드들 또는 서버들은 정상적으로 셧다운되는 이 노드들 상의 서버들에 의해, 패치를 위해 준비된다.

[0364] 단계(664)에서, 스위치 준비(prepare switch)는 패치될 노드 또는 서버에서 콜되는 바, 이는 해당 노드 또는 서버에 대한 노드 관리자가 이의 홈 디렉토리의 스위칭을 수행할 스크립트를 셋업하게 하고 동작을 수행하기 위해 요하는 파라미터들을 노드 관리자에게 제공하도록 한다.

[0365] 단계(668)에서, 노드 관리자를 재시작하기 위한 콜이 이루어지는 바, 이는 노드 관리자가 현재의 홈 디렉토리(예를 들어, 오라클 홈)를 특정된 디렉토리 경로로 이동시킬 스크립트에 제어를 전달하고, 패치된 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트 이미지를 본래의 위치에 추출하고 그 다음, 노드 관리자를 다시 시작하게 한다.

[0366] 단계(672)에서, 스위치 어서트가 실행되는 바, 이는 홈(예를 들어, 오라클 홈) 디렉토리들의 스위칭이 성공적으로 완료되었음을 확인할 것이다.

[0367] 단계(674)에서, 워크 플로우가 더 이상의 노드들 또는 서버들을 셧 다운하기 전에 패치 된 어플리케이션 서버, 어플리케이션 또는 다른 컴포넌트들 모두가 요청들을 서비스할 수 있도록 하기 위해, 시작 서버가 각 노드 또는 서버에 대해 콜되는 바, 이는 제한된 다운타임을 지원하거나 또는 다운타임이 없음 즉, 제로 다운타임을 지원한다.

[0368] * * * * * * * * * *

[0369] 제로 다운타임 패치 동안의 세션 복제

[0370] 하나의 실시예에 따르면, 제로 다운 타임 패치 동안, "제로 다운 타임"을 보장하기 위해 세션 손실로부터 보호하는 것이 중요하다. 이는, 롤링 패치 프로세스 동안의 세션 복제 및 페일오버뿐만 아니라 어플리케이션 패치로 인한 세션 호환성 문제를 고려함을 의미한다.

[0371] 전형적인 어플리케이션 서버(예를 들어, WLS) 환경에서, 시스템은 일반적으로 사용자 요청들 사이의 시간 동안 클러스터의 단 하나의 멤버가 다운되는 한, 세션이 클러스터 어딘가에서 이용 가능함을 보장하려고 한다. 주 서버가 충돌(crash)하고 그 다음, 보조 서버가 충돌하면, 세션이 손실될 수 있다. 주 서버의 모든 세션들이 단일의 보조(secondary) 서버로 복제되기 때문에, 세션 복제 분산이 클러스터 전반에서 고르게 이루어지지 않는다. 그러나 요청 페일오버는 균등하게(evenly) 분산된다. 이는, 요청들의 그룹이 다른 서버에 페일오버(fail over) 됨에 따라, 균등한 부분(even portion)이 보조 서버, 및 클러스터에 걸친 나머지 서버들의 각각에 랜딩하게 됨을 의미한다. 그 다음, 각각의 서버는 수신된 요청의 해당 부분을 담당할 것이다. 세션의 카피를 갖지 않는 서버들은 세션을 패치해야 할 것이며, 그 다음, 백업 카피를 유지할 곳을 결정하기 위해 자체의 보조 선택 알고리즘을 사용할 것이다. 오래된 카피 또는 고아 카피(orphaned copy)은 타임 아웃될 때까지 그 자리에 남아 있는다. 최종 결과적으로, 요청들의 균등 분배는 복제 알고리즘이 아니더라도 메모리 내의 세션들이 또한, 어느

정도 균등하게 분산되도록 한다.

[0372] 비동기 복제는, 요청이 완료되었지만 세션 변경들이 복제되지 않은 별개의 윈도우(distinct window)들을 가진다. 이 시간의 윈도우는 또한, 서버 충돌로 인해 요청이 페일오버되거나 프런트 엔드에서 잘못 라우팅될 때마다 서빙되는 낡은 세션(stale session)들이 존재할 수 있음을 의미한다. 하나의 실시예에 따르면, 특정한 세션 ID에 대한 세션 객체를 발견하기 위한 알고리즘은 다음과 같다. 1. 세션 ROID에 대한 로컬 맵들을 체크하고 발견되면 이를 사용한다. 2. 주 서버(primary server) 또는 보조 서버(secondary server)로부터 세션을 획득하는 것을 시도하기 위해 클라이언트 쿠키의 JVMID들을 체크한다. 3. 이용가능할 때, 해당 서버로부터 세션을 획득하고, 주 서버가 되어, 선호하는 보조 서버에 복제한다. 4. 본래의 주/보조 서버로부터의 세션이 고아가 되며 무효 또는 타임아웃시에만 정리(clean up)된다. 5. 세션이 상기로부터 이용가능하지 않으면, 새 세션을 리턴한다.

[0373] 이는, 비록 쿠키가 유효한 주 또는 보조 서버를 포인팅할 수 있지만, 세션의 로컬 카피를 이용할 가능성이 존재함을 의미한다. 이는 페일오버가 발생하고 보조 서버가 아닌 서버가 요청을 처리할 때 발생할 수 있다. 본래의 보조 서버는 낡은 카피를 가지며, 다른 페일 오버가 해당 서버에 발생하면, 이 낡은 카피는 어떤 다른 카피 전에 발견되어 사용될 것이다.

[0374] 각각의 서버에 대한 보조 선택은 자동으로 또는 선호되는 후보 서버들, 원격 후보 서버들 및 로컬 후보 서버들의 구성된 값에 기초하여 보조 서버를 선정하는 것을 시도할 것이다. 추가의 구성이 없이 자동적인 선택은 전체 서버 리스트의 현재 서버의 인덱스 및 원격 서버 목록의 사이즈의 모듈로 연산(modulo operation)에 기초하여 다른 머신으로부터 서버를 선택할 것이다. 각각의 머신이 단일 서버를 포함하고 각각의 머신이 서버들과 비슷한 순서로 조직될 때, 이는 결과적으로, 각각의 서버가 목록에서 다음 서버에 복제되게 하는 바, server1에서 server2로, server2에서 server3로, server3에서 server4로 그리고 리스트에서 서버의 마지막 서버가 server1으로 복제될 때까지 복제된다. 프런트 엔드 서버가 셧다운으로 인해 주 서버에 대한 친화성(affinity)을 유지할 수 없을 때, 이는 요청들을 나머지 클러스터화된 서버들 간에 균등한 분포로 랜덤하게 리디렉션할 것이다.

[0375] 제로 다운타임 패치 동안, 상위 계층 어플리케이션들을 포함하는 패치된 오라클 홈을 롤아웃하거나 또는 심지어는 오라클 홈 패치들과는 독립적으로 특정 어플리케이션 패치를 롤아웃하는 것이 가능하다. 이러한 어플리케이션들이 변경들을 포함할 때, 시스템은 세션 비호환성의 가능성으로부터 보호해야 한다. 세션 비호환성에 대한 일반적인 시나리오는 어플리케이션 프레임워크들의 사용으로 발생한다. 이러한 프레임워크의 새로운 버전으로 어플리케이션을 업데이트하는 것은 결과적으로, 클래스경로에 포함된 클래스에 대한 제어가 결여되게 한다. 어플리케이션 세션의 하나의 패치된 버전은 세션에 "patched.Foo" 클래스를 포함할 수 있는 반면, 어플리케이션 세션의 이전의 버전은 "unpatched.Bar" 클래스를 포함할 수 있다. 요청이 세션을 복제하려는 시도를 트리거할 때, 패치된 또는 패치되지 않은 서버에서 직렬화가 발생하고, 반대 상태의 서버에서 직렬화해제(deserialize)를 위한 시도가 발생할 수 있다. 클래스경로에 적절한 클래스들이 결여되어, 세션을 수신하는 서버는 직렬화해제 프로세스에 실패할 수 있다. 이는 결과적으로, 세션이 복제되지 않게 하고 로그 파일에 경고 메시지가 프린트되게 할 수 있다. 단일 서버에만 세션이 존재하는 경우, 이는 서버 셧다운 또는 서버 충돌에 대해 손실될 위험이 있을 수 있다.

[0376] 어플리케이션들을 패치할 때, 세션을 복제하는 능력은 중요하지만, 요청을 서빙하기 위해 세션이 일부 서버 상에서 성공적으로 직렬화해제되도록 하는 능력 역시 동일하게 중요하다. 서버가 셧다운된 후, 프런트 엔드는 요청을 균등한 분산으로 클러스터 내의 나머지 멤버들 중 하나에 랜덤하게 페일오버할 것이다. 일단, 서버가 요청을 수신하면, 이는 해당 세션의 카피를 유지하는 서버로부터 세션을 가져 오려(grap)고 시도할 것이다. 패치된 또는 패치되지 않은 서버가 반대 상태의 서버로부터 비롯된 세션을 로드하는 것을 시도할 때, 호환가능하지 않은 세션은 결과적으로, 직렬화해제 에러가 발생하게 하고 사용자는 자신의 세션 정보를 손실할 수 있다. 이러한 시나리오는, 클러스터의 다른 멤버들이 랜덤한 페일오버 요청을 처리하는 동안 서버가 셧다운되고 그 다음, 패치로 재시작되는 패치 롤아웃 프로세스 중에 종종 발생하게 된다. 이는 클러스터 멤버가 프런트 엔드 서버로부터 랜덤하게 선택될 것이기 때문에 어떤 페일오버 요청에 대한 경우일 수 있다. 또한, 느린(slow) 또는 지연(lazy) 클라이언트는 패치된 후 동일한 서버에 요청을 다시 보낼 수 있다. 이는, 패치된 서버가 일부 다른 서버 상에 저장된 "패치되지 않은 세션"을 로드하려고 시도하는 효과를 가질 것이다.

[0377] 제로 다운타임 패치는 각각의 노드를 롤링 방식으로 업데이트하는 바, 이 방식에서, server1은 셧다운되고, 패치되고 그 다음, 다음 노드로 계속하기 전에 재시작된다. 프로세스가 패치될 마지막 서버에 다다름에 따라, 오직 상기 마지막 서버 상에서 호환가능할 수 있는 패치되지 않은 서버 상에서 비롯된 세션들의 그룹이 존재한다.

이러한 세션들이 완료되기 전에 마지막 서버가 셧다운되면(타임아웃 또는 무효화), 이 세션들은 어떤 서버 상에 로드되지 않을 수 있고 손실될 것이다. 그러나, 세션들이 호환가능하면, 이들은 대기 없이 안전하게 셧다운될 수 있다.

[0378] 제로 다운타임 패치가 클러스터를 통해 롤링됨에 따라, 패치되는 서버는 자신의 주 세션들을 위험에 빠드리며 셧다운될 것이다. 이는 Server1이 셧다운될 때, 이의 세션들의 주 카피가 더 이상 이용가능하지 않기 때문이다. Server2가 보조 세션들을 호스팅하는 경우, 이들은 Server2 상의 주 상태로 승격되지만, 세션을 업데이트하기 위해 다른 요청이 들어올 때까지 세션들은 클러스터 내의 다른 서버에 복제되지 않는다. Server1을 재시작한 직후, 패치 롤아웃에서 다음 작업으로서 Server2는 셧다운될 수 있다. Server2가 셧다운되기 전에 다른 요청을 전송하지 않는 클라이언트는 자신의 세션 정보를 손실하게 된다.

[0379] 일 실시 예에 따르면, 기존 복제 서비스들에 대한 최소의 영향으로 세션 비호환성을 처리하기 위해, 패치 프레임워크는 각각의 서버에 연결되고, 일시적으로, 셧다운 시 세션들을 복제하고 이들을 패치하기 전에 고아 보조 서버들을 정리하기 위한 새로운 옵션과 함께 세션을 지연하여(lazily) 직렬화해제하기 위한 기존 옵션, 클러스터 와이드 세션 쿼리를 인에이블시킨다. 이러한 옵션들은 결합하여 세션이 클러스터에 걸쳐 적절하게 저장되고 패치 동안 세션 손실을 최소화하게 할 수 있다.

[0380] 세션 손실을 회피하는 목표를 완전히 충족키기 위해, 시스템은 세션을 로드할 수 있는 서버가 요청을 서빙하게 해야한다. 하나의 실시예에 따르면, 이는 기존 세션 처리에 대한 최소한의 방해로 다시 행해질 것이다. 서버는 긍정적으로(optimistically) 세션을 로드하는 것을 시도하고, 이를 행할 수 없는 경우, 서버는 요청을 처리할 수 있는 503 응답 코드를 사용하여 적절한 서버들의 목록을 OTD에 통신할 것이다.

[0381] 하나의 실시예에 따르면, 패치될 서버를 셧다운할 때, 세션 복제 옵션은 서버가 어떤 세션들이 보조 서버 상에서 모두 이용가능게 하는데 필수적인 이러한 세션들을 자동으로 복제하게 할 것이다. 패치 프레임워크가 클러스터에서 마지막 서버를 셧다운하려고 할 때, 이는 디폴트로, 해당 서버를 셧다운할 때 waitForAllSessions를 시그널링할 것이다. 이는 서버가 셧다운을 완료하기 전에 모든 세션들이 처리되어야 함을 웹 컨테이너에 시그널링할 것이다. 사용자는 옵션에 따라서는, 모든 어플리케이션 패치들이 호환가능한 세션 클래스들을 가지며 따라서, 클러스터 내의 마지막 서버에 대해 대기가 요구되지 않음을 시그널링하기 위한 입력을 제공할 수 있다.

[0382] 지연 세션 직렬화해제(lazy session deserialization)는 엑사로직 플랫폼들과 같은 일부 시스템들 상에서 인에이블되는 성능 기반 특징이다. ReplicatedSessionData 객체들은 세션 속성들을 직렬화해제할지를 결정 결정하기 전에 LazySessionDeserialization이 활성화는지를 체크하기 위해 ClusterMBean에 쿼리한다. 인에이블될 때, 세션 속성들은 바이트 어레이로서 효과적으로 저장될 것이다. 그러한 바이트 어레이는 나중에 속성이 검색될 때 자동으로 직렬화해제 될 것이다.

[0383] 하나의 실시예에 따르면, 필요할 때만 세션을 로딩하는 이러한 능력의 장점을 취하기 위해, 기능성이 동적으로 만들어질 수 있다. 패치 프레임워크는 패치 프로세스 중 지연 세션 직렬화해제를 인에이블/디스에이블하는 역할을 가질 것이다. 이는 또한, 구성 값이기 때문에, ClusterMBean 구성이 이미 인에이블되지 않은 경우 프레임워크는 오직 설정을 변경하는 것을 시도한다. 그렇지 않으면 각 피관리 서버 상의 ClusterService는 런타임 값들을 수신하는 데 사용될 것인 바, 이 값은 인에이블 시, 구성된 값들보다 우선할 것이다. 이는, 심지어 LazyDeserialization가 오프된 때에도 ClusterService가 이를 턴 온할 수 있음을 의미한다. 그러나, 이는 사용자가 이를 온으로 구성했을 때에는 디스에이블시킬 수 없다. 이 값은 런타임 값일 것이기 때문에, 패치 프레임워크는 ClusterService를 복수 번 호출해야 할 것이다. 제1 통지는 클러스터 내의 어떤 서버들이 패치되기 전에 발생한다. 이는 ClusterService 상에서 LazySessionDeserialization을 설정하기 위해 RemoteClusterServicesOperations 인터페이스를 사용하여 클러스터 내의 각각의 서버에 연결될 것이다. 제2 통지는 서버가 패치 및 재시작된 후에 발생할 것이다. 재시작 후에, 서버는 다시 구성된 값을 사용하므로 LazySessionDeserialization을 인에이블하기 위해 런타임 설정을 재확립하는 것이 필요할 것이다. 패치 프로세스가 완료되면, 패치 프레임워크는 필요에 따라 지연지 세션 직렬화해제 옵션을 디스에이블할 것이다.

[0384] 하나의 실시예에 따르면, 패치 프레임워크는 서버들의 리스트의 쌍의 포맷으로, 서버들의 현재 상태에 관하여 클러스터의 각각의 멤버에게 알림(alert)할 것이다. 서버 이름들의 하나의 리스트는 그룹으로 간주될 것이고, 서버 이름들의 다른 리스트는 다른 그룹으로 간주될 것이다. 다시, 통지들이 필요한 두 가지 다른 포인트가 존재할 것이다. 제1 통지는 서버를 셧다운시키고 패치를 적용한 후에 발생될 것이다. 해당 서버를 재시작하기 전에, 클러스터는 패치된 리스트에 합류하는 새로 패치된 서버를 갖는 새로운 그룹을 통지받을 것이다. 이는, 패치된 서버가 재시작됨에 따라 실행중인 서버들이 낡은 정보를 가지지 않도록 할 것이다. 제2 통지는 서버가 시

작된 직후에 발생할 것이며, 프레임워크는 모든 어플리케이션들이 준비될 때까지 대기한다. 목표는 서버가 세션 비호환성을 수반하는 요청을 정확하게 처리 할 수 있도록 가능한 빨리 상태를 통지받게 하는 것이다. 마지막으로 패치 프로세스가 완료된 후, 값들은 클러스터에게로의 최종 통지와 함께 null로 리셋될 것이다. 이는, 패치 전의 상태를 복원하여서 클러스터는 더 이상 패치가 진행 중이라고 가정하지 않으므로 거동은 다시 디폴트로 리턴될 수 있다.

[0385] 하나의 실시예에 따르면, 웹 컨테이너는 긍정적으로, 복제된 세션을 검색는 것을 시도할 것이다. 직렬화해제 오류가 발생하면, 웹 컨테이너는 현재의 서버 그룹들을 체크할 것이다. 현재의 서버 그룹들의 값은 패치가 현재 진행 중인지 여부를 나타낼 것이다. 웹 컨테이너는 현재의 서버가 속한 그룹을 식별하기 위해 그룹들의 컨텐츠들을 검사할 것이다. 현재의 서버 이름을 포함하지 않는 그룹은 현재의 서버가 호환가능하지 않고 따라서 다른 그룹이 호환가능해야 한다는 논리에 기초하여 호환가능한 그룹으로 간주될 것이다. 이는, 순방향(forward) 및 역(backward) 호환성 문제를 모두 서빙해야 한다. 일단, 웹 컨테이너가 세션이 가장 호환될 가능성이 큰 서버들의 그룹을 식별하면, 이는 해당 그룹의 서버들의 리스트를 갖는 "X-WebLogic-Cluster-FailoverGroup-List" 헤더와 함께 503 응답 코드를 반환할 것이다.

[0386] 하나의 실시예에 따르면, OTD는 서버 그룹을 포함하는 헤더와 함께 503을 수신 할 것이고, 요청을 리디렉션하기 위해 그 리스트로부터 서버들을 랜덤으로 선택할 것이다. OTD는 이것이 WLS가 가지지 않은 정보이므로 드레인 풀 내의 서버들을 처리해야 한다. 서버가 특정한 리스트(server-specified list)는 런타임시 생성된 클러스터 내의 현재의 멤버들을 포함할 것이다. 이는 클러스터에 합류하는 웹로직 서버의 동적인 발견과 유사하게 프론트엔드에 의해 처리되어야 한다. 리스트는 본질적으로 동적이며 런타임 중에 변경될 수 있지만, 리스트는 패치 프로세스 시작시 알려진 모든 클러스터 멤버들을 포함할 것이다.

[0387] 하나의 실시예에 따르면, 패치 프레임워크는 패치 동안 세션들의 적절한 처리를 가능하게 하는 역할을 가질 것이다. 셧다운 동안의 이러한 세션들의 복제는 클러스터 와이드 세션 쿼리 및 고아 보조 서버 정리를 모두 인에이블시키는 것에 좌우된다. 프레임워크는 오직, ClusterMBean 구성이 해당 설정을 인에이블하지 않은 경우에 어떤 설정을 변경하는 것을 시도할 것이다. 프레임워크는 패치 전에 각각의 서버에 연결되고 각각의 플래그를 인에이블시킬 것이다. 그 다음, 각각의 서버가 재시작됨에 따라, 플래그들은 다시 설정되어야 할 것이다. 마지막으로, 패치 프로세스가 완료된 후, 필요에 따라 설정이 복귀될 것이다.

[0388] 하나의 실시예에 따르면, WLS-MT 클러스터링을 위해 이미 구현된 세션 페치는 클라이언트 쿠키를 업데이트함이 없이 세션을 보조 서버에 자동으로 복제하기 위해 이용되어서, 페일오버 요청이 클러스터의 어떤 멤버 상에 랜딩하게 되고, 세션을 발견하기 위한 어떤 메커니즘이 필요하게 될 것이다. 요청이 서버에 랜딩할 때의 거동은 다음과 같다. 1. 세션 ROID에 대해 로컬 맵을 체크하고, 발견되면 이를 사용한다. 2. 주 서버 또는 보조 서버로부터 세션을 획득하는 것을 시도하기 위해 클라이언트 쿠키의 JVMID를 체크한다. 3. 이용가능한 경우, 해당 서버로부터 세션을 획득하고, 주 서버가 되어, 선호되는 보조 서버에 복제한다. 4. 본래의 주/보조 서버 상의 고아 세션들을 처리하기 위한 새로운 메커니즘이 도입될 것이다. 5. 세션이 상기로부터 이용가능하지 않은 경우, 다음을 수행한다. SessionFetching이 인에이블되지 않으면, 새 세션을 리턴한다. SessionFetching이 인에이블된 경우, 브로드캐스트 쿼리를 클러스터에 전송한다. 제1 응답은 세션을 획득할 수 있는 서버를 식별하는 데 사용될 것이다. 주 서버가 되어 선호되는 보조 서버에 복제한다. ii. 본래의 주/보조 서버 상의 고아 세션들을 처리하기 위한 새로운 메커니즘이 도입될 것이다.

[0389] 하나의 실시예에 따르면, 서버 셧다운 동안, 다른 클러스터 멤버들에게 셧다운을 통지하기 직전에, ReplicationService는 상기 세션의 각각의 주 카피가 보조 서버에 복제되도록 할 것이다. 이는, 서버의 셧다운 동작 동안 세션이 손실되지 않게 할 것이다. 이는 오직, 본래의 주 서버가 재시작된 이래로 요청을 하지 않은 (이는 클라이언트들이 새로운 보조 서버로 새로운 주 서버를 재확립하지 않음을 의미함) 클라이언트들에게만 영향을 끼칠 것이다. 마지막으로, 그러한 클라이언트가 리턴할 대, 세션은 클러스터 내의 일부 서버 상에서 이용가능할 것이다.

[0390] 하나의 실시예에 따르면, 고아 세션들은 셧다운 또는 세션 페치 시 세션 복제에 고유하지 않다. 그러나 각각의 서버가 연속적으로 재시작되는 클러스터의 반복(iteration)으로 인해 이 문제는 발생할 가능성이 더욱 높다.

[0391] 고아 보조 서버로부터의 낡은 세션 데이터를 서빙할 확률을 처리하기 위해, 패치 후에 고아 보조 카피들을 정리하기 위한 메커니즘이 존재할 것이다. 패치 동안 이 기능이 인에이블되면, ReplicationService는 해당 세션을 패치한 후 고아 세션들의 정리를 처리할 백그라운드 프로세스를 트리거할 것이다. 백그라운드 프로세스는 세션 버전 번호, 세션이 발견된 타임스템프 정보, 세션과 관련되었을 수 있는 어떤 다른 서버, 및 새로운 보조 서버

를 알고 있을 것이다. 이는, 세션의 현재의 카피들 제거함이 없이 버전 및 타임스탬프 정보에 기초하여 모든 낡은 카피들을 정리할 수 있게 할 것이다.

[0392] 하나의 실시예에 따르면, 서버가 정상적으로 셧다운될 때, 사용자는 웹 컨테이너가 복제되지 않은 세션들의 완료를 기다리게 하기 위해 ignoreSessions = false를 특정할 수 있다. 그러나, 웹 컨테이너는 클러스터 어딘가에 세션 복제본 존재하기 때문에 복제된 세션을 기다리지 않을 것이다. 그러나, ZDT 패치의 경우, 세션이 호환가능하지 않고 서버가 클러스터의 마지막 패치되지 않은 서버인 경우, 서버는 호환가능한 세션들을 가진 유일한 서버가 되며, 이는 모든 세션들이 완료 될 때까지 기다려야 한다. 정상적인 셧다운을 위한 "waitForAllSessions" 플래그가 이러한 목적으로 도입된다.

[0393] 패치 프레임워크는 디폴트로, 클러스터의 마지막 서버에서 셧다운을 콜할 때 "waitForAllSessions" 부울(Boolean)을 특정할 것이다. 이는, 셧다운 시퀀스를 완료하기 전에 모든 세션들이 무효화되는 것을 대기하도록 웹 컨테이너에 시그널링할 것이다. 관련된 세션이 없는 모든 요청들이 503 응답에 의해 거부될 것이고, OTD는 503 응답을 받으면 이러한 요청들을 서빙하기 위해 클러스터 내의 다른 서버들을 시도할 것이다. 기존 세션들을 갖는 모든 요청들이 적절하게 서빙될 것이다. 웹 컨테이너는 세션들이 패치된 서버들 중 어느 것에서 호환가능하지 않을 수 있기 때문에 완료될 때까지 이 세션들 각각을 처리해야 한다.

[0394] 사용자는 옵션에 따라서는, waitForAllSessions가 거짓(false)일 수 있음을 시그널링하기 위해, 패치 동작을 시작할 때 SessionCompatibility = true를 특정할 수 있다. waitForAllSessions 옵션은 기존 ignoreSessions 파라미터와 유사한 ServerLifeCycleRuntimeMBean에 추가된다. 다양한 실시예들에 따르면, 추가적인 파라미터들이 지원될 수 있는데, 예를 들어, 패치를 위해 다음 피관리 서버를 셧다운하기 시작하기 전에 얼마나 오래 기다리는지를 나타내는 타임 아웃 (delayBetweenNodes)이 지원될 수 있는 바, 이는 보조 세션들이 서버를 셧다운시키기 전에 복제되게 하는 데 유용할 수 있다.

[0395] * * * * * * * * * *

[0396] 퀵 스타트 예시(Quick Start Example)

[0397] 하나의 실시예에 따르면, 제로 다운타임 패치는, 한 번에 하나의 노드에 변경들을 롤아웃하고, 트래픽 디렉터(예를 들어, OTD)가 변경이 완료될 때까지 유입 트래픽을 나머지 노드에 리다이렉트하게 함으로써 달성될 수 있다. 예를 들어, 오라클 홈의 패치를 위한 전형적인 시퀀스 동작들은 다음을 포함한다. 1. 관리자는 패치를 검증한다. 2. 오라클 홈 및 대표 도메인의 카피가 생성된다. 3. 패치는 테스트/검증 환경에 적용된다. 4. 테스트는 패치가 프로덕션을 위해 승인되도록 하기 위해 시행된다. 5. 검증된 오라클 홈은 스크립트를 사용하여 카피되며, 생성된 아카이브는 프로덕션 환경에 걸쳐 롤아웃될 패치된 "Gold Master"로 간주된다. 6. 생성된 오라클 홈 아카이브는 어드미니스트레이터에 의해 프로덕션 환경에 걸쳐 각각의 물리적 머신에 배포된다. 7. 어드미니스트레이터는 롤아웃 동작을 실행한다.

[0398] JavaHome의 인스톨레이션/업데이트, 및 어플리케이션 소스들의 분배는 마찬가지로 이들 롤아웃 동작들에 대한 어드미니스트레이터에게 달려 있을 수 있다. 하나의 실시예에 따르면, 타겟 환경은 어드민 서버를 실행할 하나의 노드를 포함하여 3 개 이상의 물리적 머신들 또는 노드들을 포함해야 한다. 하나의 실시예에 따르면, 추가적인 요건들은 피관리 서버들이 제로 다운타임을 지원하기 위해 클러스터에 존재해야 한다는 것과, 각각의 노드는 이 서버가 어드민 서버를 실행시키는 것을 포함하여 자신의 노드 관리자가 실행되게 한다는 것과, 오라클 홈 디렉토리는 각각의 노드에, 바람직하게는 모든 노드 상에서 동일한 위치(예를 들어, /scratch/aime1/OracleHomes/wls1221)에 로컬하게 인스톨되어야 한다는 것과, 그리고 도메인 디렉토리는 오라클 홈 디렉토리의 외부에 존재해야 한다는 것을 포함한다.

[0399] 어드미니스트레이터는 오라클 홈의 아카이브 jar을 생성하기 위해 이동 스크립트의 장점을 취하고 이 아카이브 jar을 각각의 원격 노드에 카피함으로써 모든 노드에서 인스톨 및 도메인을 복제해야 하는 것을 회피할 수 있다.

[0400] 하나의 실시예에 따르면, 도메인은 적어도 2 개의 피관리 서버 및 적어도 3 개의 노드 관리자를 참조해야 한다. 도메인은, 도메인의 카피를 만들고, 원격 노드들 중 둘 모두에 해당 바이너리를 배포하고 그 다음, 각각의 원격 노드에서 언팩을 수행하는 것을 포함하여 팩/언팩 유틸리티들을 사용하여 복수의 노드들에 대해 복제될 수 있다.

[0401] JavaHome 롤아웃이 성공하기 위해서는, 새로운 JavaHome이 영향을 받는 각 머신 상에 인스톨되어야 하며, 각각의 머신 상의 동일한 경로에 위치되어야 한다. 이는, 현재의 노드 관리자 및 피관리 서버들이 실행되는 동안 행

해져야 해서, 인스톨은 기존 JavaHome 경로를 변경시키지 않아야 한다. 이를 돕기 위해, JavaHome은 symlink들을 포함하는 경로가 아닌 절대적 경로로서 특정되어야 한다.

[0402] 일단, 롤아웃 동작이 시작되면, 오라클 홈에 대한 어떤 변경들은 한 번에 하나의 노드에 적용될 것이다. 어드미니스트레이터는 요구되는 패치들을 적용하기 위해 아래에 더 기술되는 바와 같이, OPatch 툴을 사용할 수 있다. 일부 커스토머들은 Puppet 또는 Chef와 같은 파일의 배포를 도울 수 있는 준비된 툴들을 가질 수 있다.

[0403] * * * * * * * * * *

[0404] OPatch와의 통합

[0405] 하나의 실시예에 따르면, 시스템은 예를 들어 오라클 제품의 범위에 걸쳐 제로 다운타임 패치를 위한 커스토머-대면 프론트 엔드를 제공하기 위해 OpatchAuto와 같은 제품과 통합될 수 있다. 이러한 피쳐(feature)들을 통합시키는 것은 단일 인터페이스 하에서 더욱 완벽한 솔루션을 제공한다.

[0406] 하나의 실시예에 따르면, OPatchAuto는 사용자가 예를 들어, WLS 컴포넌트들의 패치된 버전을 생성하고, 이들이 업데이트될 노드들에 액세스가능하게 하고, 그리고 패치 롤아웃을 인보크 및 모니터링 할 수 있게 하는 툴을 제공한다. 패치 인프라스트럭쳐는 서버들의 런타임 상태들 및 가용성을 관리하고, WLS 컴포넌트들 및 어플리케이션 소스들을 업데이트하며, 어떤 멀티-테넌시 문제를 해결하면서도, 활성 세션들이 보존되게 한다.

[0407] 일부 상황들에서, 커스토머는 비프로덕션 환경에서 검증 테스트를 수행하기 위해 롤아웃으로부터 패치된 아카이브들의 생성을 분리하길 원할 수 있거나 또는, 이들은 이 부분들을 결합하는 단일 동작을 원할 수 있다. 하나의 실시예에 따르면, OPatchAuto는 패치된 WLS 아카이브를 생성하고, 모든 노드들에 아카이브를 이용가능하게 하며, 그리고 별개의 또는 결합된 단계들로서 롤아웃을 개시하기 위한 능력을 제공한다. 사용자는 모든 노드에 배포될 패치된 바이너리를 생성하고, 모든 노드 상에서 패치된 바이너리를 스테이지하고, (WLS가 런타임 관리 및 롤아웃을 담당하게 하여) 서비스 다운타임이 없이 패치된 바이너리의 런타임 활성화를 수행하기 위해 OPatchAuto를 사용할 수 있다.

[0408] 하나의 실시예에 따르면, OpatchAuto는, ZDT 패치가 토폴로지에 대해 지원되는지 여부를 결정할 수 있도록 패치 메타데이터를 검사하기 위한 능력을 제공하고, 테스트를 위해 오프라인 패치 환경을 생성하는 워크플로우 기능을 제공하는 것을 포함하여, WLS 환경에서 제로 다운타임 패치를 구동하는 엔트리 포인트로서 역할을 한다. 이는 프로덕션 환경으로부터 직접 또는 프로덕션 환경과 균등한 것으로 추정되는 기존 오라클 홈을 카피하기 위한 능력을 포함한다.

[0409] 추가적으로, OPatchAuto는 성공적으로 패치되고 테스트된 오라클 홈 아카이브를 토폴로지의 다양한 노드에 분배하는 워크플로우 능력을 제공할 것이다. 이는, 언제든지 OPatchAuto로 개시될 수 있는 환경이 롤아웃을 위해 준비되게 할 것이다. OPatchAuto는 또한, 패치 롤아웃들을 개시하고 모니터링하기 위해 이용될 수 있다.

[0410] 패치 인프라스트럭쳐는 서버들이 업데이트될 순서를 결정하고, 패치 롤아웃의 단계들을 모니터링하고 진행할 때와 및 필요한 경우 복귀시킬 때를 결정하고, 세션이 보존되게 하고, 서버 수명 주기를 관리하고 패치된 오라클 홈 비트들에서 스와핑하고, 상태 업데이트들을 위해 OPatchAuto에 의해 쿼리될 표준 진행 객체를 제공하고, 그리고 어떤 서버들이 패치될 것이고 어떤 서버들이 패치되었는지에 관한 정보를 제공하기 위해 진행 객체를 강화(enhance)시키는 역할을 한다. 이 정보는 또한, 롤아웃이 실행을 시작하기 전에 진행 객체를 통해 이용가능해질 것이다.

[0411] 예시

[0412] 어플리케이션 서버(예컨대, WLS) 도메인은 MW_HOME 외부에 생성된다. OPatchAuto 월렛이 SSH/JMX을 통해 호스트에 연결하기 위해 생성된다.

```
- . ./common.sh
- ./config-wallet.sh -create "${USER}&${HOSTNAME}:ssh"
  "${USER}&${HOST1}:ssh" "${USER}&${HOST2}:ssh"
  "${USERNAME}&${HOSTNAME}:wls"
```

[0413]

[0414] 패치를 어드민 서버에 적용하고 패치된 오라클 홈 아웃-오브-플레이스에 기초하여 아카이브를 생성한다.

```
- ${ORACLE_HOME}/OPatch/auto/core/bin/opatchauto.sh apply
  ${PATCH_HOME} -create-image -image-location
  ${WLSZDT_DIR}/image.jar -oop [-oh
  /path/to/different/oraclehome]
```

[0415]

[0416] 검증 후, 업데이트될 노드들 전부에 패치된 아카이브를 스테이지시킨다.

```
- ${ORACLE_HOME}/OPatch/auto/core/bin/opatchauto.sh apply -
  plan wls-push-image -image-location ${WLSZDT_DIR}/image.jar
  -wls-admin-host ${HOSTNAME}:7001 -wls-target Cluster1 -
  remote-image-location
  ${WLSZDT_DIR}/rolloutOracleHomeImage.jar -wallet
  ${WALLET_DIR} [-walletPassword passswordIfNeeded]
```

[0417]

[0418] 전체 도메인 또는 특정한 클러스터에 롤아웃을 개시하고 모니터링한다.

```
- ${ORACLE_HOME}/OPatch/auto/core/bin/opatchauto.sh apply -
  plan wls-zdt-rollout -image-location ${WLSZDT_DIR}/image.jar
  -wls-admin-host ${HOSTNAME}:7001 -wls-target Cluster1 -
  backup-home ${WLSZDT_DIR}/home-backup -remote-image-location
  ${WLSZDT_DIR}/rolloutOracleHomeImage.jar -wallet
  ${WALLET_DIR} [-walletPassword passswordIfNeeded]
```

[0419]

[0420] 실패된 롤아웃들을 재개 또는 롤백한다.

```
- ${ORACLE_HOME}/OPatch/auto/core/bin/opatchauto.sh resume -
  session SEID [-walletPassword passwordIfNeeded]

- ${ORACLE_HOME}/OPatch/auto/core/bin/opatchauto.sh rollback -
  session SEID [-walletPassword passswordIfNeeded]
```

[0421]

[0422] 본 발명은 본 발명의 교시들에 따라 프로그램된 하나 이상의 프로세서들, 메모리 및/또는 컴퓨터 판독가능 저장 매체들을 포함하여 하나 이상의 종래의 범용 또는 특수용 디지털 컴퓨터, 컴퓨팅 디바이스, 머신 또는 마이크로프로세서를 이용하여 편리하게 구현될 수 있다. 적절한 소프트웨어 코딩이 소프트웨어 분야의 숙련자들에게 분명할 바와 같이 본 발명의 교시들에 기초하여 숙련된 프로그래머들에 의해 쉽게 준비될 수 있다.

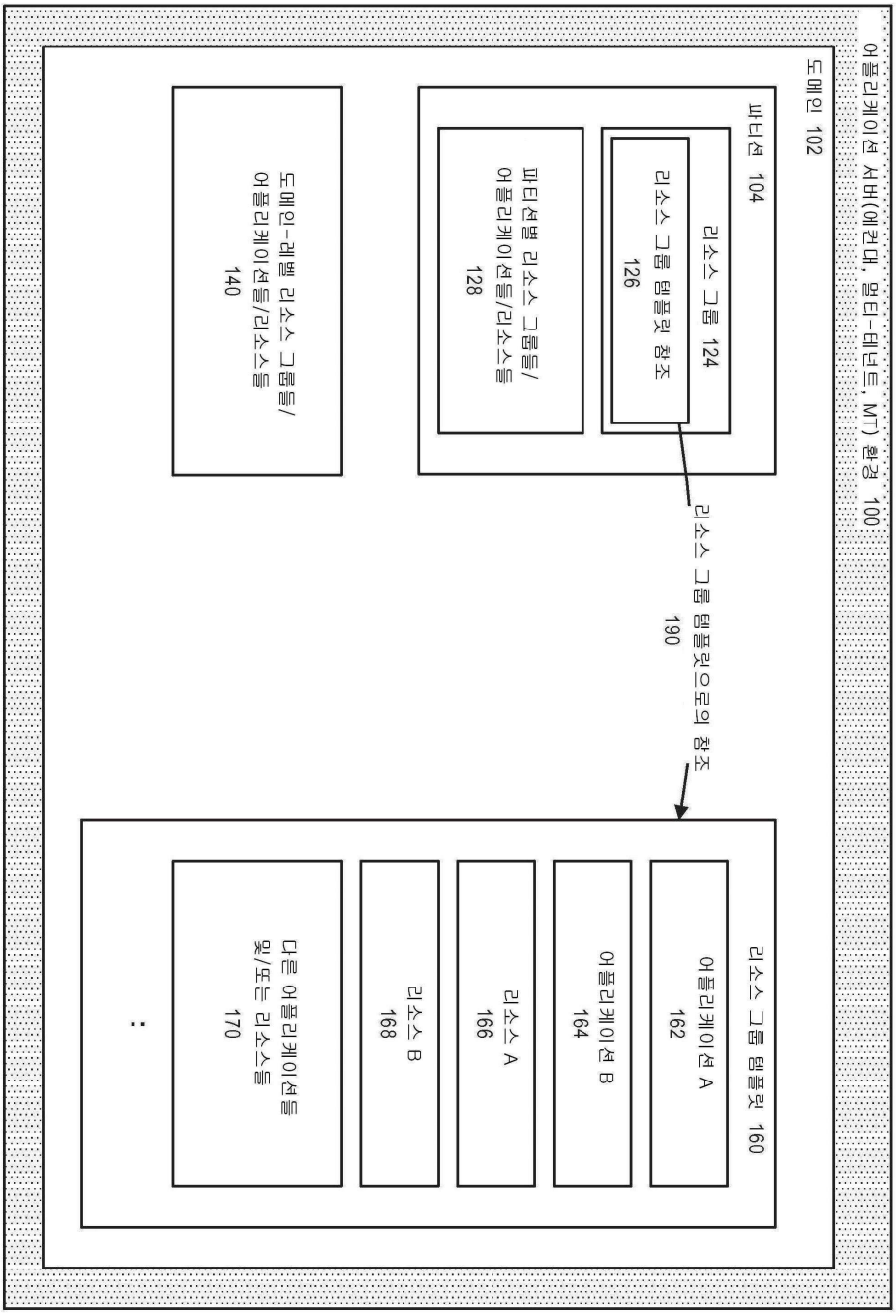
[0423] 일부 실시예들에서, 본 발명은 컴퓨터 프로그램 물을 포함하는 바, 상기 컴퓨터 프로그램 물은 명령어들이 저장된/본 발명의 프로세스들 중 어느 것을 수행하도록 컴퓨터를 프로그래밍하기 위해 이용될 수 있는 비일시적 저장 매체 또는 컴퓨터 판독가능 매체(매체들)이다. 저장 매체는 이들로만 한정되는 것은 아니지만, 플로피 디스크(disk)들, 광학 디스크(disc)들, DVD, CD-ROM들, 마이크로드라이브 및 자기-광학 디스크(disk)들을 포함하는 어떤 타입의 디스크, ROM들, RAM들, EPROM들, EEPROM들, DRAM들, VRAM들, 플래시 메모리 디바이스들, 자기 또는 광학 카드들, (분자 메모리 IC들을 포함하는)나노시스템들 또는, 명령어들 및/또는 데이터를 저장하기에 적절한 어떤 타입의 매체 또는 디바이스를 포함할 수 있다.

[0424] 본 발명의 상기 설명은 예시 및 설명을 목적으로 제공되었다. 본 설명은 완전한 것(exhaustive)으로 의도되거나 정확히 개시된 형태들로만 본 발명을 제한하고자 의도된 것이 아니다. 많은 수정들 및 변형들이 이 기술분야의 숙련자에게 분명할 것이다. 위 실시예들은 본 발명의 원리 및 이의 실용적 응용을 가장 잘 설명하기 위해 선택 및 기술되었으며, 그럼으로써 이 기술분야의 숙련자들은 본 발명에 대한 다양한 실시예들 및 고려되는 특별한 사용에 적합한 다양한 수정들을 이해할 수 있다. 본 발명의 범위는 다음의 특허 청구 범위 및 이의 균등물에 의

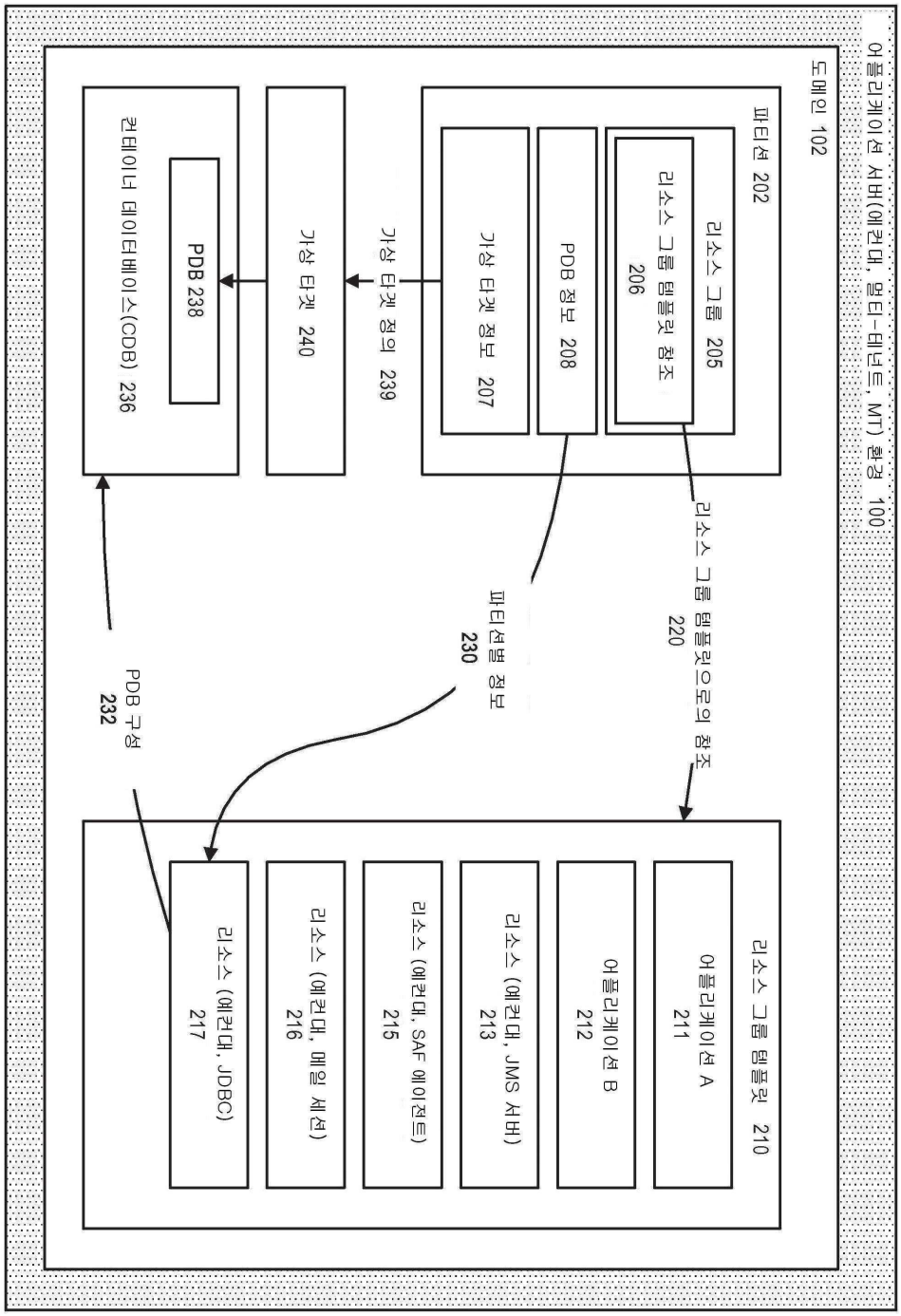
해 한정되어야 함이 의도된다.

도면

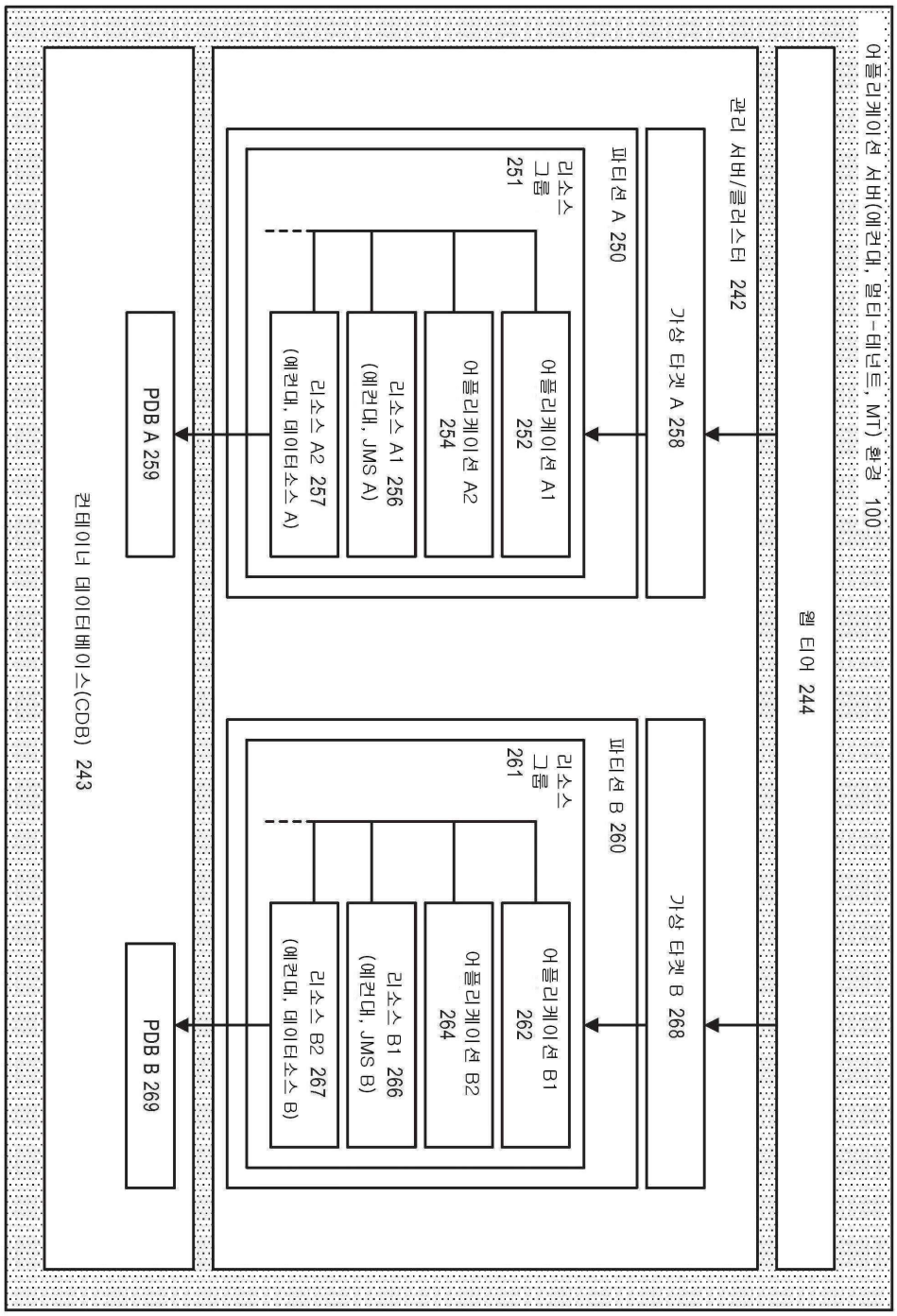
도면1



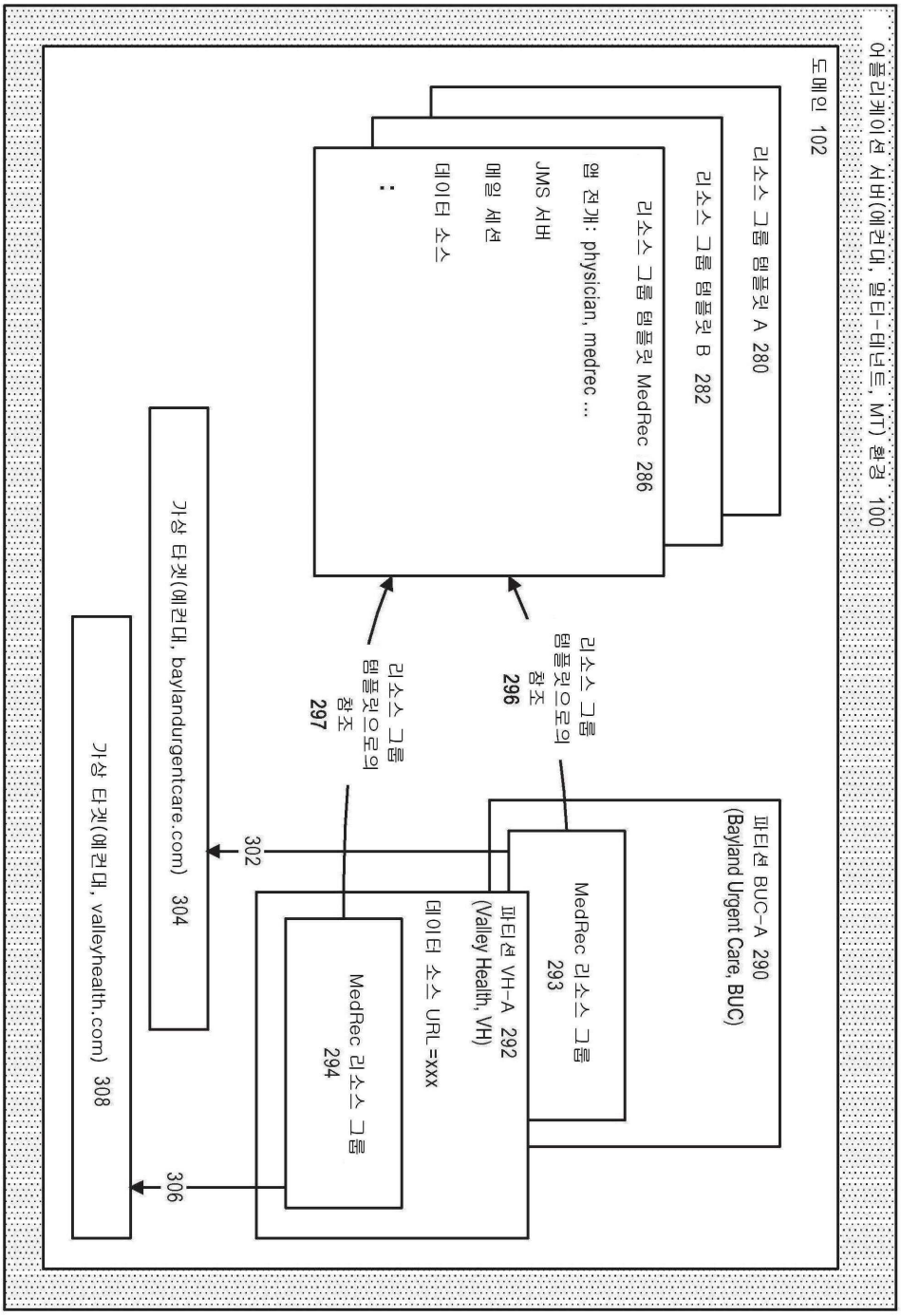
도면2


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
파티션 202
리소스 그룹 205
리소스 그룹 템플릿 참조 206
PDB 정보 208
가상 타겟 정보 207
가상 타겟 정의 239
가상 타겟 240
PDB 238
컨테이너 데이터베이스(CDB) 236
리소스 그룹 템플릿 210
어플리케이션 A 211
어플리케이션 B 212
리소스 (예컨대, JMS 서버) 213
리소스 (예컨대, SAF 에이전트) 215
리소스 (예컨대, 메일 세션) 216
리소스 (예컨대, JDBC) 217
리소스 그룹 템플릿으로의 참조 220
파티션별 정보 230
PDB 구성 232

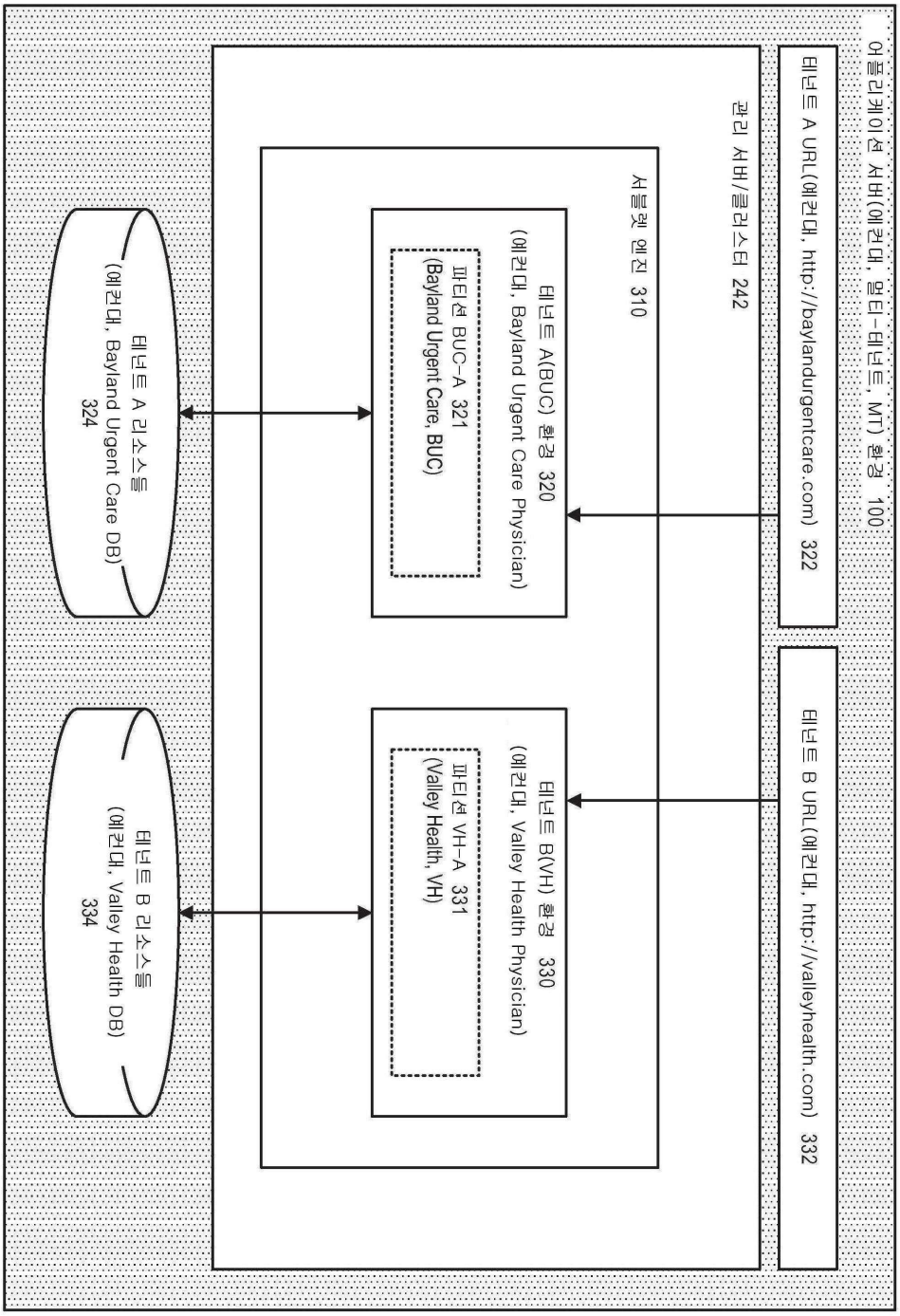
도면3


어플리케이션 서버(에퀀대, 멀티-테넌트, MT) 환경 100
웹 티어 244
관리 서버/클러스터 242
가상 타겟 A 258
파티션 A 250
리소스 그룹 251
어플리케이션 A1 252
어플리케이션 A2 254
리소스 A1 256 (예퀀대, JMS A)
리소스 A2 257 (예퀀대, 데이터소스 A)
가상 타겟 B 268
파티션 B 260
리소스 그룹 261
어플리케이션 B1 262
어플리케이션 B2 264
리소스 B1 266 (예퀀대, JMS B)
리소스 B2 267 (예퀀대, 데이터소스 B)
PDB A 259
PDB B 269
컨테이너 데이터베이스(CDB) 243

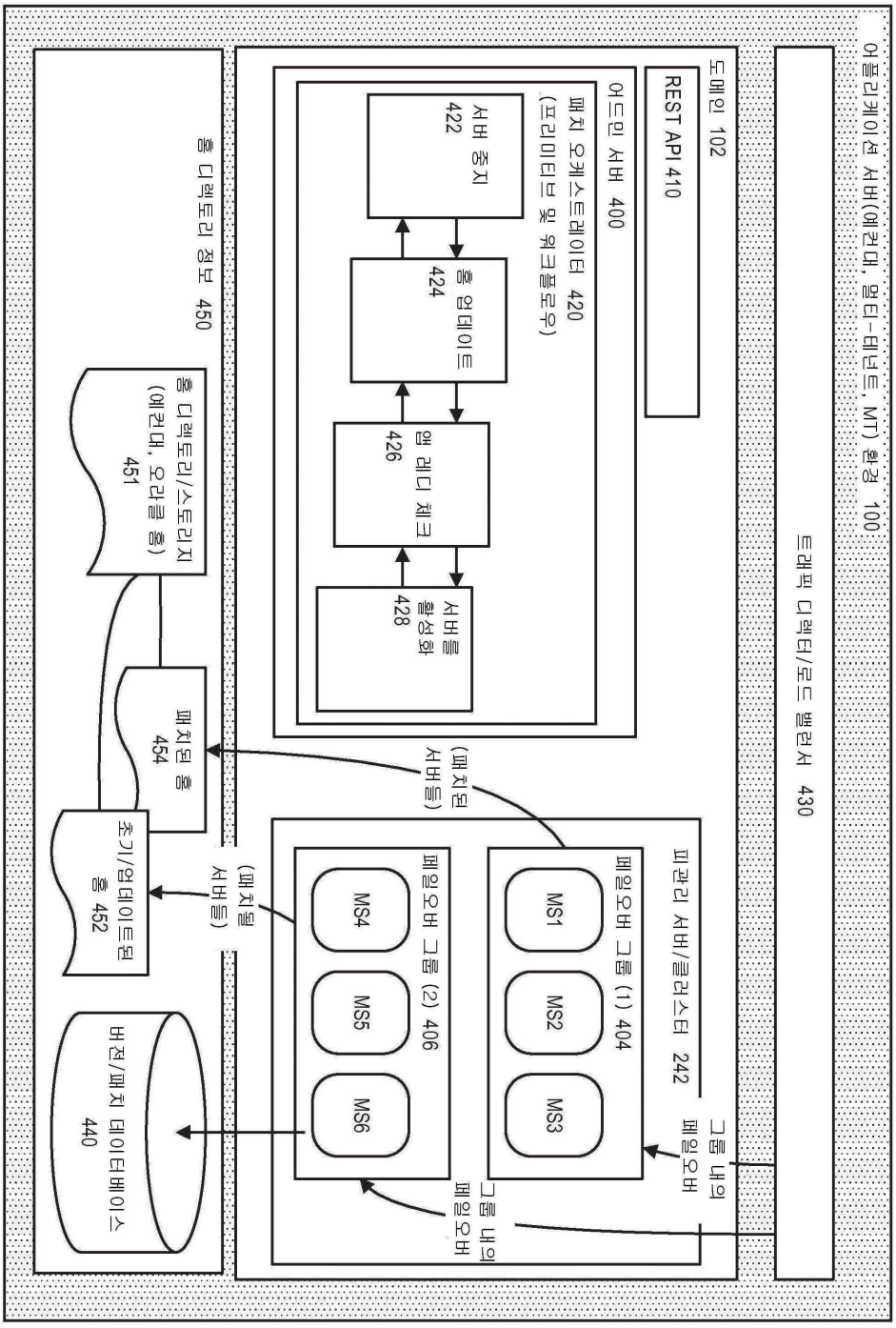
도면4


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
리소스 그룹 템플릿 A 280
리소스 그룹 템플릿 B 282
리소스 그룹 템플릿 MedRec 286
앱 전개: physician, medrec ...
JMS 서버
메일 세션
데이터 소스
:
리소스 그룹 템플릿으로의 참조 296
리소스 그룹 템플릿으로의 참조 297
파티션 BUC-A 290
(Bayland Urgent Care, BUC)
MedRec 리소스 그룹 293
파티션 VH-A 292
(Valley Health, VH)
데이터 소스 URL=xxx
MedRec 리소스 그룹 294
302
306
가상 타겟(예컨대, baylandurgentcare.com) 304
가상 타겟(예컨대, valleyhealth.com) 308

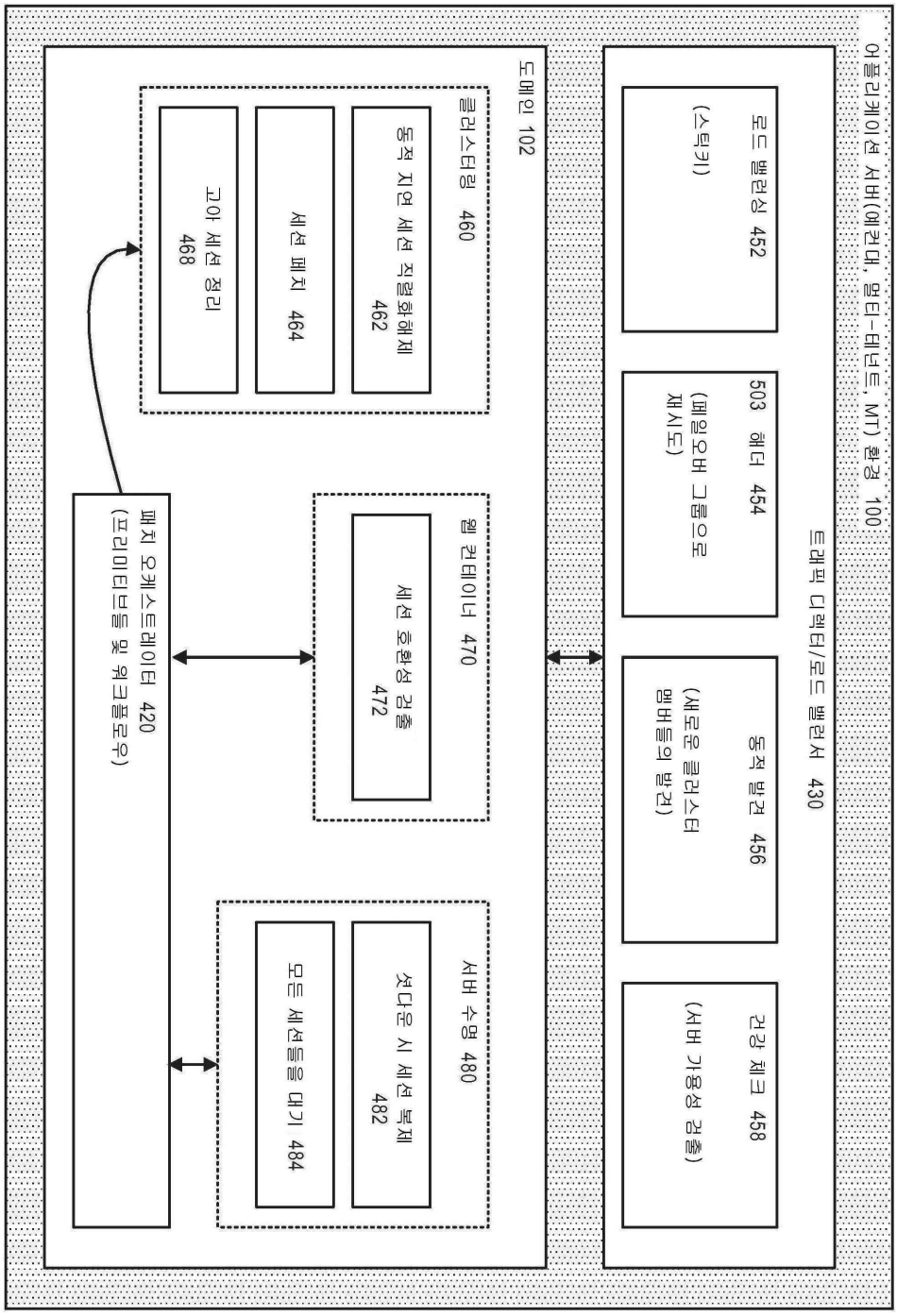
도면5


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
테넌트 A URL(예컨대, http://baylandurgentcare.com) 322
테넌트 B URL(예컨대, http://valleyhealth.com) 332
관리 서버/클러스터 242
서블렛 엔진 310
테넌트 A(BUC) 환경 320
(예컨대, Bayland Urgent Care Physician)
파티션 BUC-A 321
(Bayland Urgent Care, BUC)
테넌트 B(VH) 환경 330
(예컨대, Valley Health Physician)
파티션 VH-A 331
(Valley Health, VH)
테넌트 A 리소스들
(예컨대, Bayland Urgent Care DB)
324
테넌트 B 리소스들
(예컨대, Valley Health DB)
334

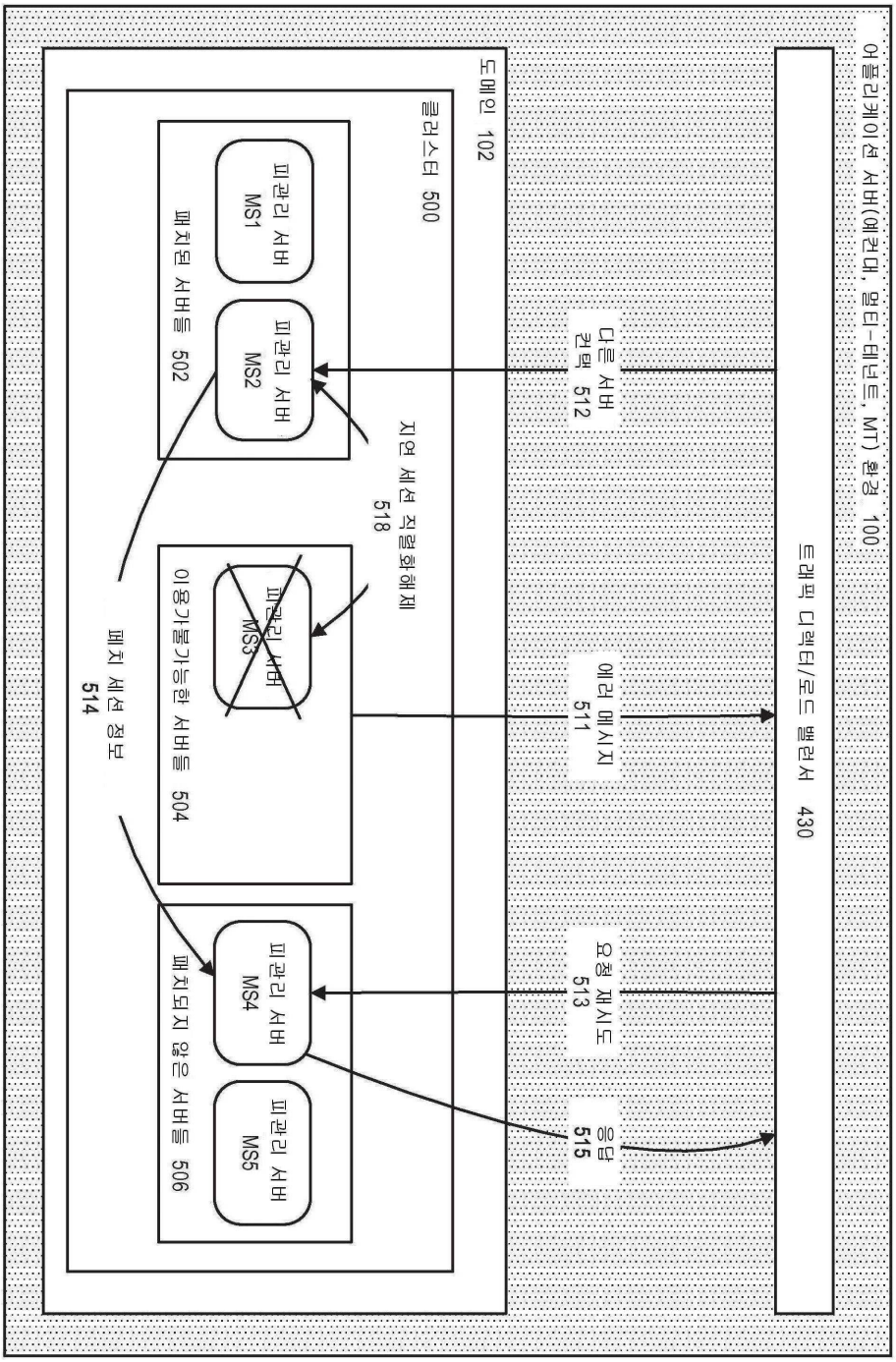
도면6


어플리케이션 서버(에퀀대, 멀티-테넌트, MT) 환경 100
트래픽 디렉터/로드 밸런서 430
도메인 102
REST API 410
어드민 서버 400
패치 오케스트레이터 420
(프리미티브 및 워크플로우)
서버 중지
422
홈 업데이트
424
앱 레디 체크
426
서버를 활성화
428
피관리 서버/클러스터 242
그룹 내의 페일오버
페일오버 그룹 (1) 404
MS1
MS2
MS3
그룹 내의 페일오버
페일오버 그룹 (2) 406
MS4
MS5
MS6
(패치된 서버들)
(패치될 서버들)
홈 디렉토리 정보 450
홈 디렉토리/스토리지
(에퀀대, 오라클 홈)
451
패치된 홈
454
초기/업데이트된 홈
452
버전/패치 데이터베이스
440

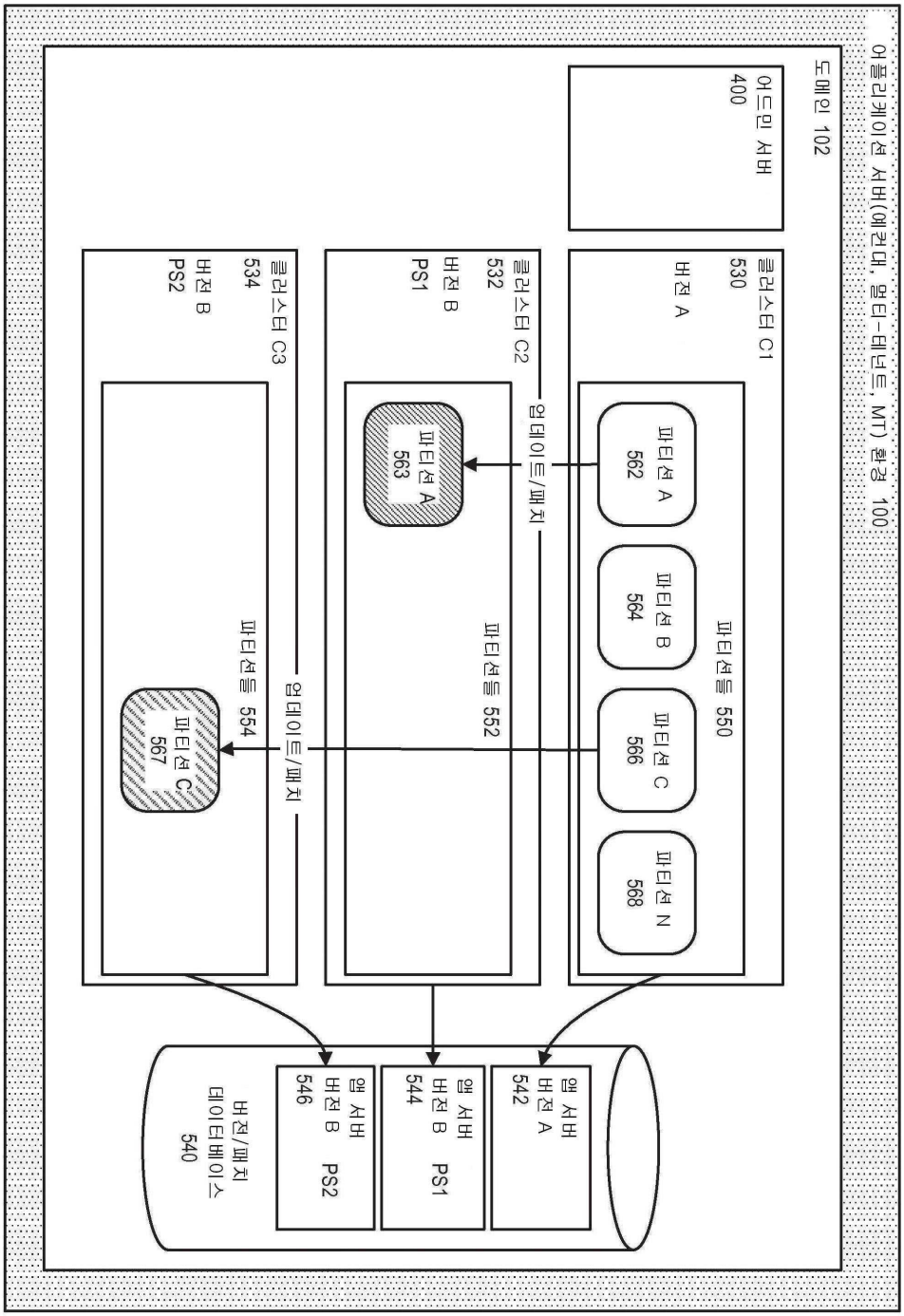
도면7


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
트래픽 디렉터/로드 밸런서 430
로드 밸런싱 452
(스틱키)
503 헤더 454
(페일오버 그룹으로 재시도)
동적 발견 456
(새로운 클러스터 멤버들의 발견)
건강 체크 458
(서버 가용성 검출)
도메인 102
클러스터링 460
동적 지연 세션 직렬화해제 462
세션 패치 464
고아 세션 정리 468
웜 컨테이너 470
세션 호환성 검출 472
서버 수명 480
셧다운 시 세션 복제 482
모든 세션들을 대기 484
패치 오케스트레이터 420
(프리미티브들 및 워크플로우)

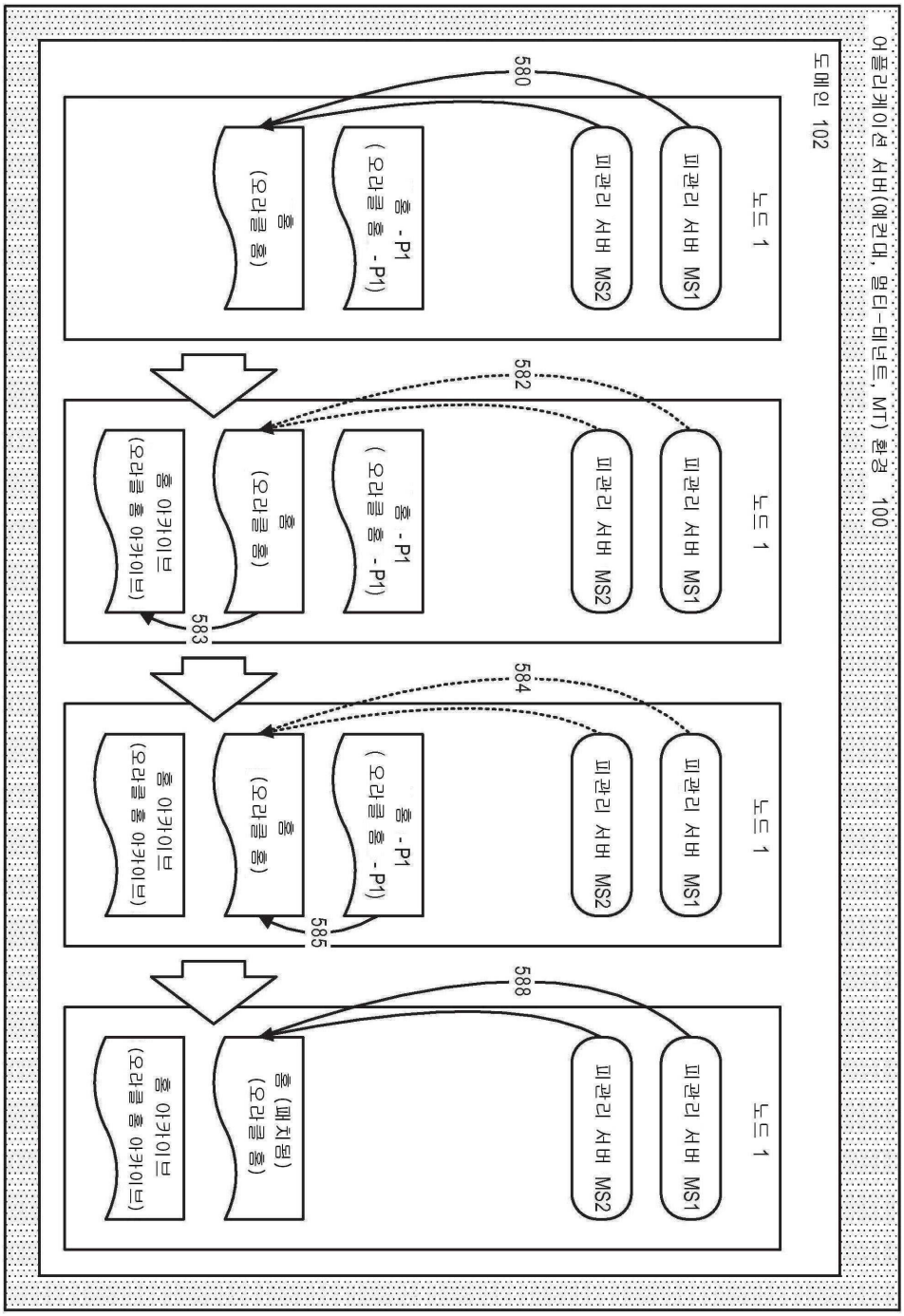
도면8


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
트래픽 디렉터/로드 밸런서 430
다른 서버 컨택 512
에러 메시지 511
요청 재시도 513
응답 515
도메인 102
클러스터 500
지연 세션 직렬화 해제 518
패치 세션 정보 514
패치된 서버들 502
피관리 서버 MS1
피관리 서버 MS2
이용가능한 서버들 504
피관리 서버 MS3
패치되지 않은 서버들 506
피관리 서버 MS4
피관리 서버 MS5

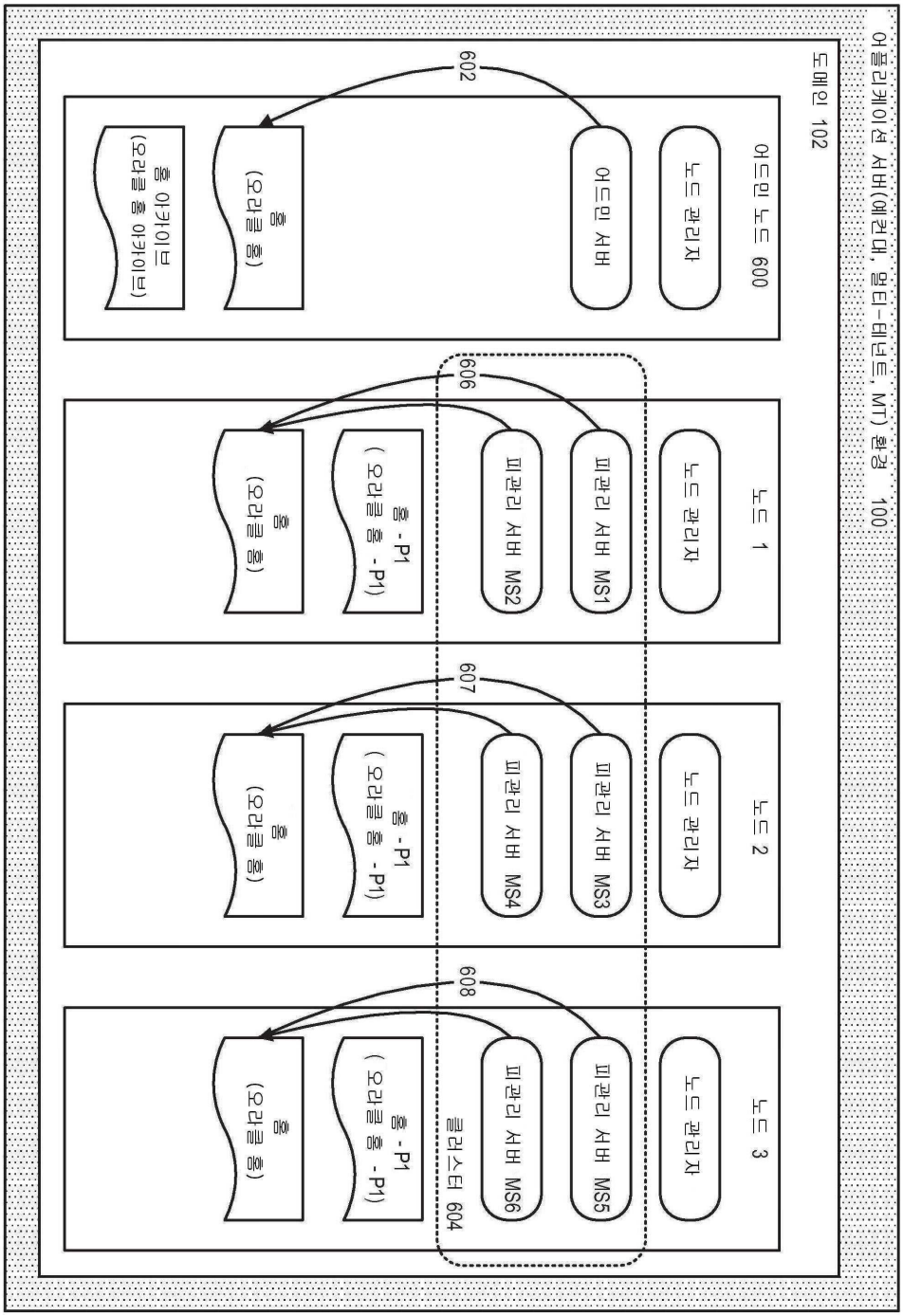
도면9


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
어드민 서버
400
클러스터 C1
530
버전 A
파티션들 550
파티션 A
562
파티션 B
564
파티션 C
566
파티션 N
568
업데이트/패치
클러스터 C2
532
버전 B
PS1
파티션들 552
파티션 A'
563
클러스터 C3
534
버전 B
PS2
파티션들 554
파티션 C'
567
업데이트/패치
앱 서버
버전 A
542
앱 서버
버전 B PS1
544
앱 서버
버전 B PS2
546
버전/패치
데이터베이스
540

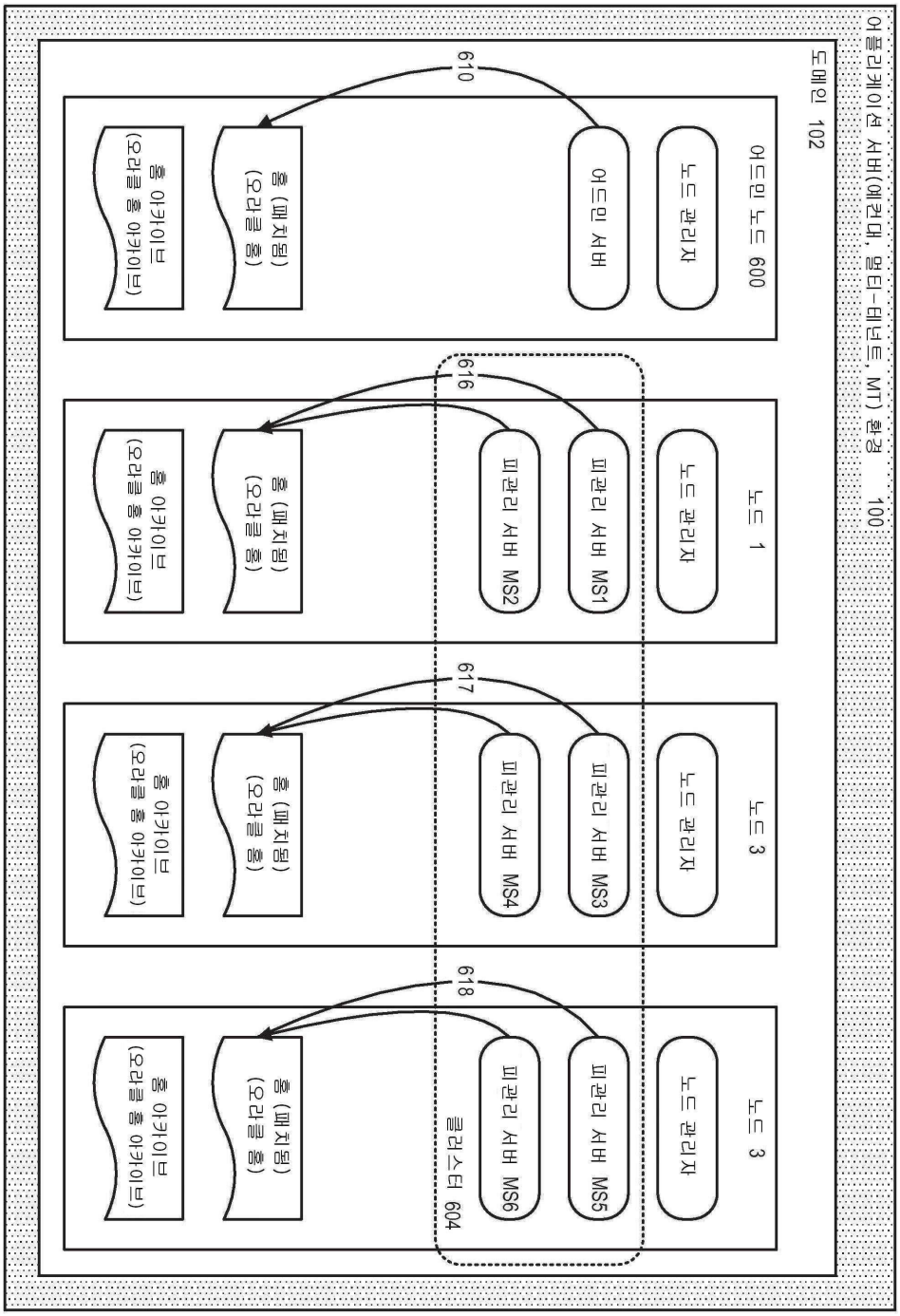
도면10


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
노드 1
피관리 서버 MS1
피관리 서버 MS2
풀 - P1 (오라클 풀 - P1)
풀 (오라클 풀)
풀 데이터 (오라클 풀 데이터)
풀 (패치됨) (오라클 풀)
580
582
583
584
585
588

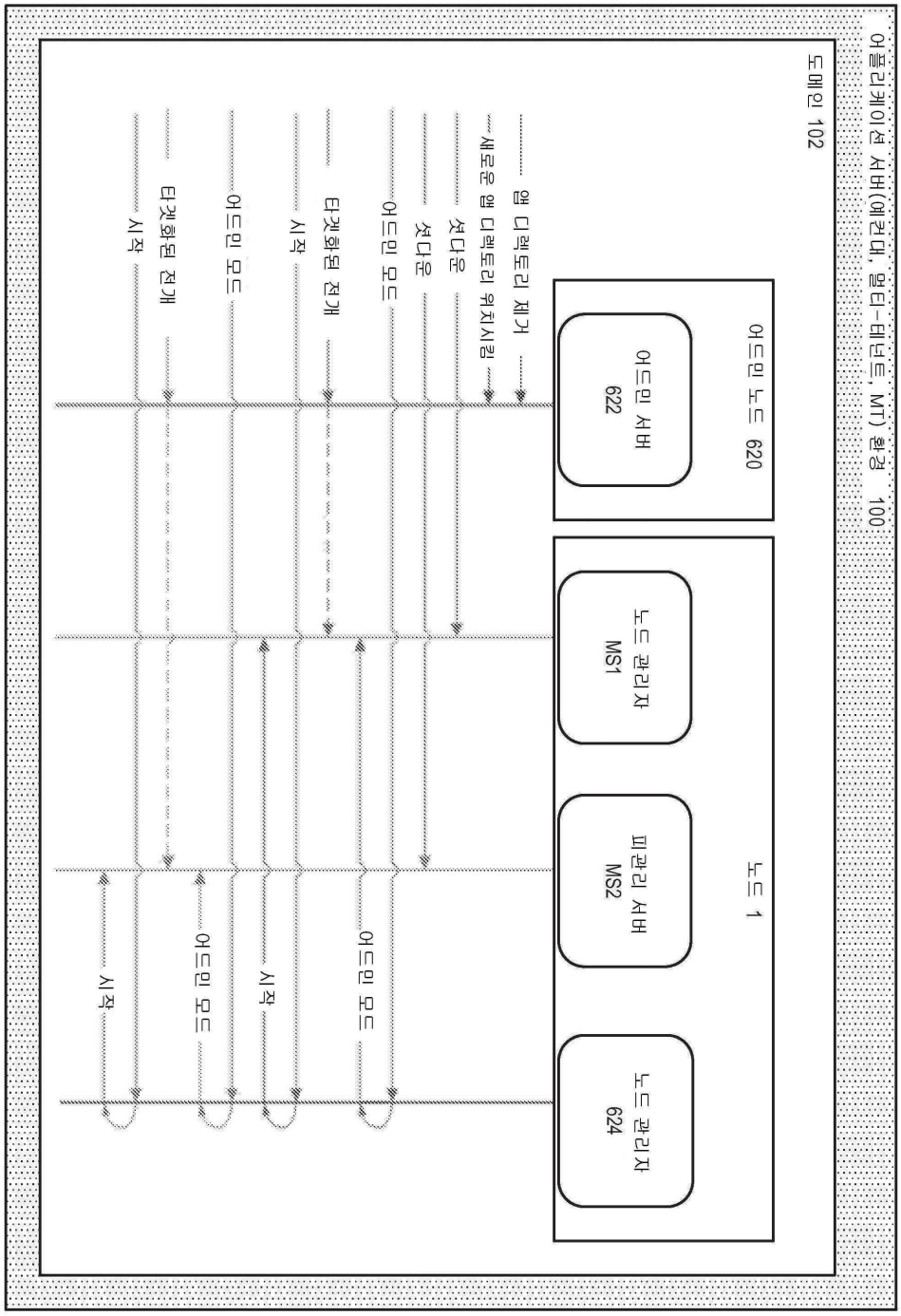
도면11


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
어드민 노드 600
602
노드 관리자
어드민 서버
메타데이터 (오라클 메타데이터)
노드 1
606
노드 관리자
피관리 서버 MS1
피관리 서버 MS2
메타데이터 - P1 (오라클 메타데이터 - P1)
메타데이터 (오라클 메타데이터)
노드 2
607
노드 관리자
피관리 서버 MS3
피관리 서버 MS4
메타데이터 - P1 (오라클 메타데이터 - P1)
메타데이터 (오라클 메타데이터)
노드 3
608
노드 관리자
피관리 서버 MS5
피관리 서버 MS6
메타데이터 - P1 (오라클 메타데이터 - P1)
메타데이터 (오라클 메타데이터)
클러스터 604

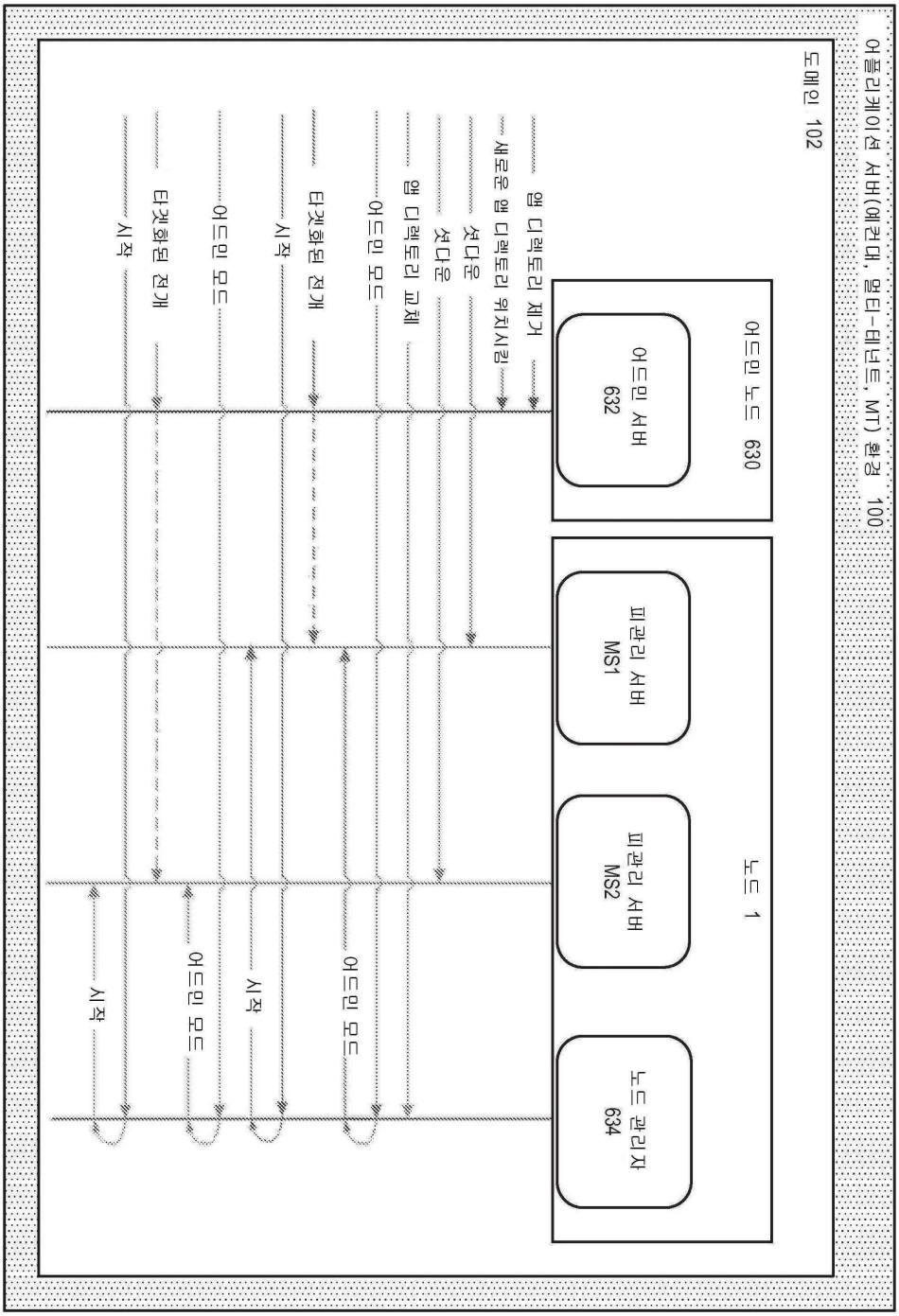
도면12


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
어드민 노드 600
노드 관리자
어드민 서버
610
홈 (패치됨)
(오라클홈)
(유저아너10 홈 롤라오)
유저아너10 홈
노드 1
노드 관리자
피관리 서버 MS1
피관리 서버 MS2
616
노드 3
노드 관리자
피관리 서버 MS3
피관리 서버 MS4
617
노드 3
노드 관리자
피관리 서버 MS5
피관리 서버 MS6
618
클러스터 604

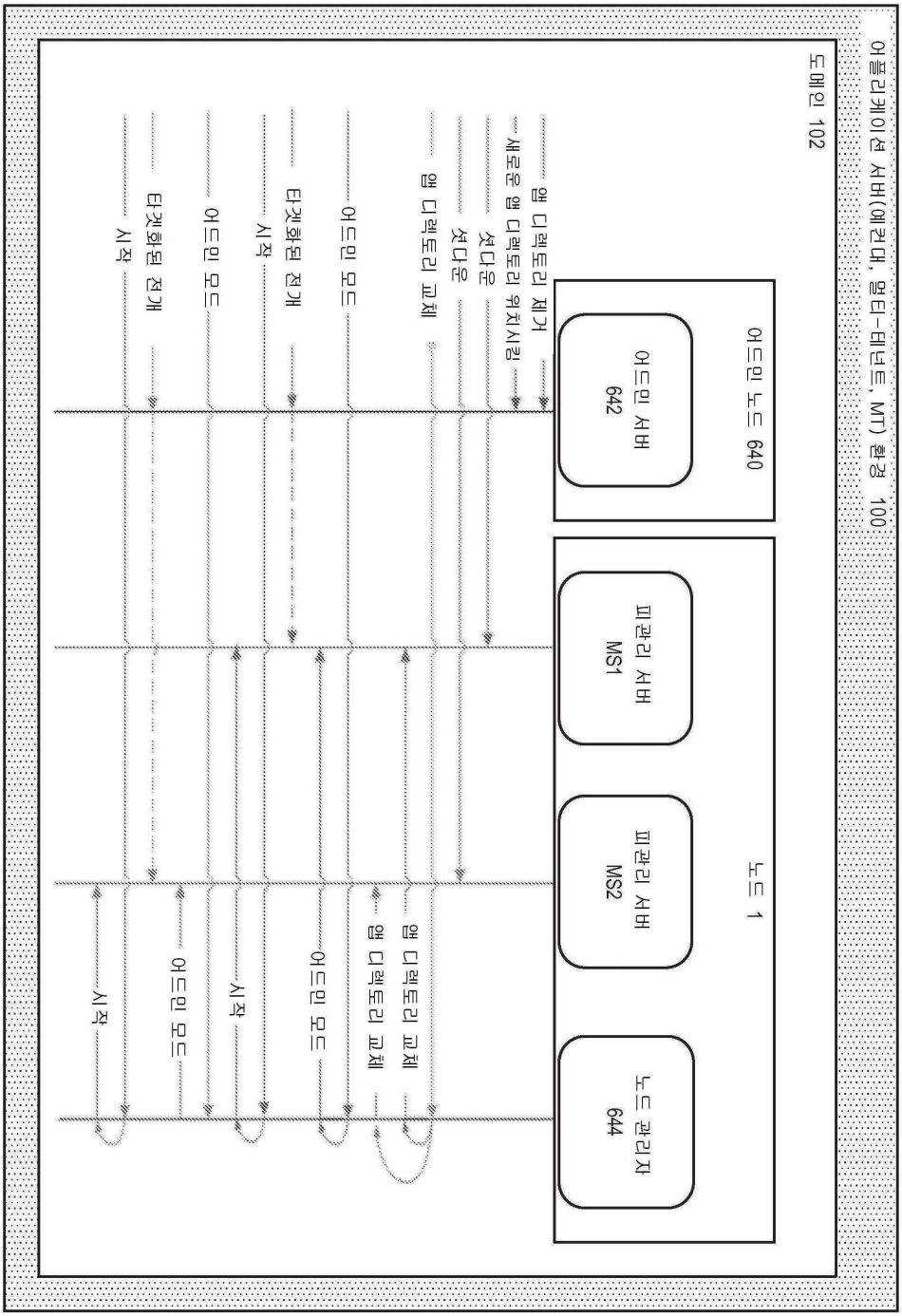
도면13


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
어드민 노드 620
어드민 서버 622
노드 1
노드 관리자 MS1
피관리 서버 MS2
노드 관리자 624
앱 디렉토리 제거
새로운 앱 디렉토리 위치시킴
셧다운
어드민 모드
타겟화된 전개
시작
어드민 모드
타겟화된 전개
시작
어드민 모드
시작
어드민 모드
시작

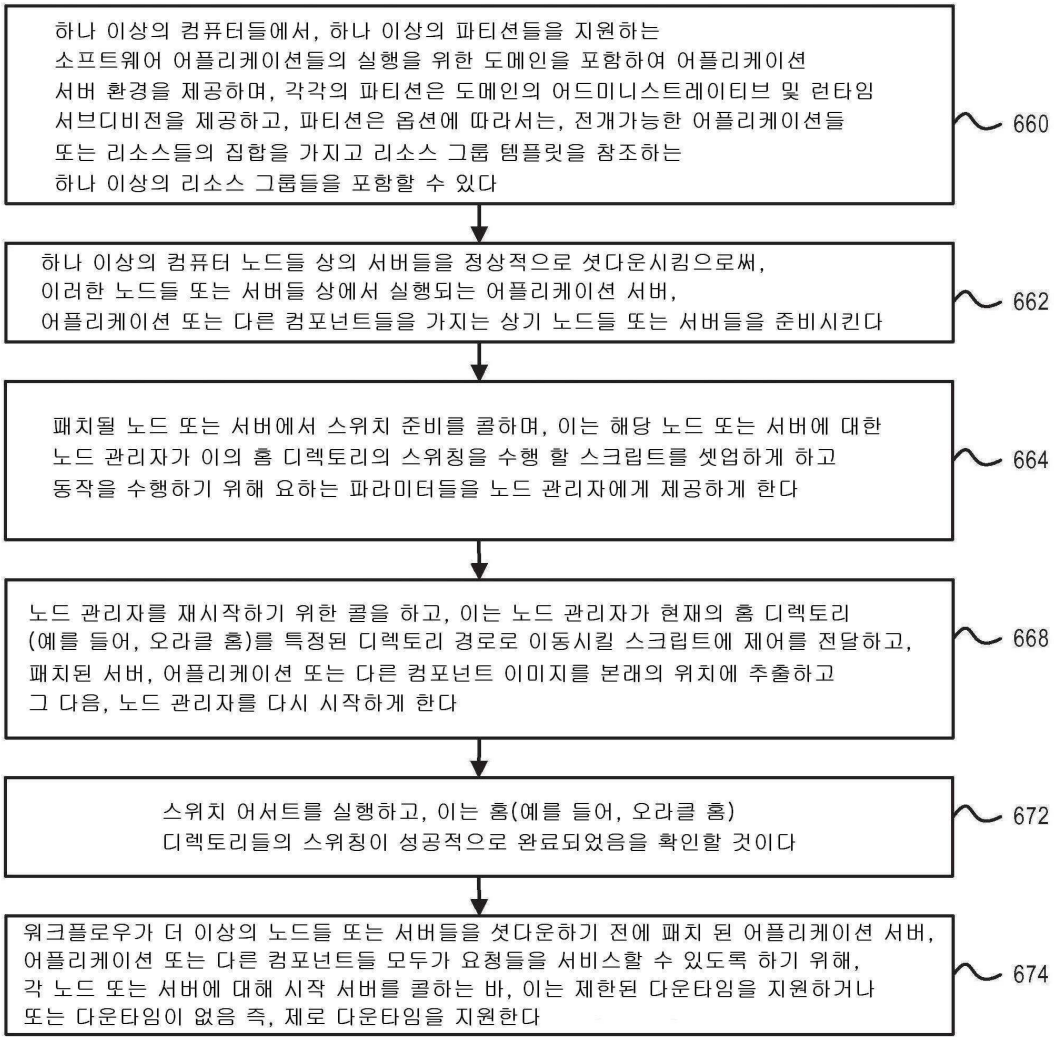
도면14


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
어드민 노드 630
어드민 서버 632
노드 1
피관리 서버 MS1
피관리 서버 MS2
노드 관리자 634
웹 디렉토리 제거
새로운 앱 디렉토리 위치시킴
셧다운
셧다운
앱 디렉토리 교체
어드민 모드
타겟화된 전개
시작
어드민 모드
타겟화된 전개
시작
어드민 모드
시작
어드민 모드
시작

도면15


어플리케이션 서버(예컨대, 멀티-테넌트, MT) 환경 100
도메인 102
어드민 노드 640
어드민 서버 642
노드 1
피관리 서버 MS1
피관리 서버 MS2
노드 관리자 644
앱 디렉토리 제거
새로운 앱 디렉토리 워치시킴
셧다운
앱 디렉토리 교체
어드민 모드
타겟화된 전개
시작
어드민 모드
타겟화된 전개
시작
앱 디렉토리 교체
앱 디렉토리 교체
어드민 모드
시작
어드민 모드
시작

도면16



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 19

【변경전】

하나 이상의 컴퓨터 시스템들 상에서 실행되기 위한 명령어들을 포함하는 컴퓨터 판독가능한 저장 매체에 포함된 컴퓨터 프로그램으로서, 상기 명령어들은 실행될 때, 상기 하나 이상의 컴퓨터 시스템들이 청구항 제7항 내지 제12항의 방법을 수행하도록 하는 것을 특징으로 하는 컴퓨터 프로그램.

【변경후】

하나 이상의 컴퓨터 시스템들 상에서 실행되기 위한 명령어들을 포함하는 컴퓨터 판독가능한 저장 매체에 포함된 컴퓨터 프로그램으로서, 상기 명령어들은 실행될 때, 상기 하나 이상의 컴퓨터 시스템들이 청구항 제7항 내지 제12항 중 어느 한 항의 방법을 수행하도록 하는 것을 특징으로 하는 컴퓨터 프로그램.