부록 B: 한국어판 특별 부록

클라우드 파운드리 환경의 준비와 활용

글쓴이: 정윤진

이 책에서는 주로 클라우드와 같이 분산 시스템을 운영하기 좋은 환경 기반에서 효율적으로 서비스를 운영하기 위한 방법에 대한 내용을 소개했다. 이를 위해서는 애플리케이션뿐만 아니라 이 애플리케이션을 구동하는 인프라스트럭처 구조 역시 12요소 지원을 위해 변화되어야 하기에, 이 구조를 분산에 적합하게 구현하는 방법에 대해 주로 다뤘다. 또한 책에서 소개되는 내용의 이해를 돕기 위해 저자들은 다양한 예제를 준비했다. 대부분의 예제가 개발자의 랩탑이나 데스크탑 환경에서도 무리없이 동작하겠지만, 간혹 클라우드 파운드리 환경이 필요한 경우가 있다. 특히 4부에서는 클라우드 환경에서 스프링 애플리케이션을 배포하고, 배포된 애플리케이션을 클라우드 파운드리상에서 운영하고 처리하는 방법에 대해 자세하게 설명한다. 하지만 정작 클라우드 파운드리를 준비하는 방법, 그리고 애플리케이션을 배포하는 방법, 서비스를 준비하는 방법에 대해서는 자세히 설명해놓지 않았기에 국내 독자의 이해를 돕기 위해 이에 대한 내용을 정리해 한국어판 특별 부록으로 추가한다. 직접 실행해보고 실제로 클라우드 파운드리를 사용함으로써 개발자들이 어떤 혜택을 누릴 수 있는지 저자들의 설명과 함께 직접 느껴 보기를 바란다.

클라우드 파운드리의 소개

클라우드 파운드리Cloud Foundry는 VMware의 프로젝트 B29로 명명된 도구로 데렉 콜리슨Derek Collison에 의해 디자인되고 개발되었다. 비슷한 시기에 유사한 이름으로 아마존 EC2 기반의 자바로 만들어진 플랫폼 서비스PaaS가 있었지만, 오늘날 클라우드 파운드리와는 다르다.

최근 들어 많은 회사에서 도입해 사용 중인 클라우드 파운드리는 2011년 4월경에 발표된 것이다. 일 년 뒤에는 대규모의 애플리케이션의 배포와 수명 주기를 관리하는 보쉬BOSH(http://bosh.io)가 발표되었으며, 클라우드 파운드리는 이후 보쉬를 통해 다양한 클라우드 서비스에 배포되었다. 보쉬는 구글이 자사의 서비스 배포와 수명 주기를 관리하는 데 사용한다고 알려진 보그BORG에 기반을 두고 만들어졌으며, 처음에는 다소 학습이 어려울 수 있지만 이후에는 현존하는 스크립트 기반의 도구들에 비해 배포된 소프트웨어와 인프라를 관리할 수 있는 기능이 탁월한 강력한 도구가 되었다.

2013년 EMC, VMware를 모회사로 하는 피보탈 소프트웨어Pivotal Software는 클라우드 파운드리와 래빗엠큐, 그리고 스프링 관련된 제품들을 개발하기 시작했다. 2015년에 클라우드 파운드리는 클라우드 파운드리 파운데이션CFF, Cloud Foundry Foundation(http://cloudfoundry.org)을 통해 플랫폼 생태계를 확장했다. 8개의 회사로 시작한 이 클라우드 파운드리 생태계에는 현재 50여 개가 넘는 대형 회사들이 참여하고 있으며, 대한민국의 NIA에서도 파스타 프로젝트를 통해 가입하고 있다.

클라우드 파운드리의 목적은 애플리케이션과 지원 서비스의 손쉬운 배포 방법을 개발팀에 제공해 개발팀이 자신이 만든 소프트웨어를 직접 운용할 수 있는 능력을 제공하는 데 있다. 이는 '만들면 운영까지.'라는 아마존이나 넷플릭스와 같은 성공적인 회사에서 주창하는 클라우드 기반의 문화에도 깊이 닿아 있다. 한번 풀었던 어려운 운영의 문제를 매번 반복하지 않고, 최대한 자동화를 구현해서 필요한 도구와 함께 개발팀에 운영 편의를 제공하는 것을 목적으로 한다. 2018년에 이르러 클라우드 파운드리는 클라우드 파운드리 2.0으로 진화하면서 애플리케이션 플랫폼과 컨테이너 플랫폼, 두 개로 확장되었다. 이중 컨테이너 플랫폼은 보쉬를 통해 쿠버네티스Kubernetes를 다양한 클라우드 서비스에 배포하는 프로젝트다.

다양한 회사들이 클라우드 파운드리 파운데이션의 멤버로 활동하고 있으며, 대부분은 피보탈 클라우드 파운드리의 고객이기도 하다. 이들 중 IBM의 블루믹스, 알리 클라우드, SAP, 오라클

등 자사의 공용 클라우드 서비스를 공개한 업체들은 클라우드 파운드리를 가져다가 자사의 플랫폼 서비스로 제공하고 있기도 하다. 피보탈은 클라우드 파운드리 코드의 90% 이상에 기여하며 오픈소스 클라우드 파운드리와 이를 바탕으로 한 피보탈 클라우드 파운드리를 엔터프라이즈에 공급하고 있다.

스프링과 클라우드 파운드리, 래빗엠큐, 아파치 톰캣 등 다양한 오픈소스에 기여하고 있는 피보탈은 2018년 4월 뉴욕 증권 시장에 상장했으며, 포춘 100대 기업 중 절반 이상을 고객사로 둔 회사로 발돋움 하고 있다.

클라우드 파운드리를 활용한 좀 더 손쉬운 환경구축

이 책에서는 굉장히 많은 스프링 프로젝트를 언급하고 있다. 동시에 다양한 예제를 함께 소개하고 있는데, 이들 중 일부는 지원 서비스와 연동이 필요하다. 예를 들어 9장의 데이터 부분에서 JPA를 사용하려면 MySQL, 레디스, 네오포제이 같은 서비스 인스턴스가 필요하며, 10장에서 메시징 구현의 동작 확인을 위해서는 래빗엠큐나 카프카 인스턴스가 필요할 것이다. 이러한 서비스 인스턴스들은 직접 설치해서 사용할 수도 있지만, 책에 소개된 바와 같이 피보탈 클라우드 파운드리를 쓰면 훨씬 더 간편하게 애플리케이션을 배포해서 사용할 수 있다. 특히 4부에서 소개되는 서비스 브로커와 같은 내용들을 실습해 보려면 클라우드 파운드리의 준비가 필수적이라고 할 수 있다.

책에 소개되는 내용을 더 쉽게 활용하기 위해, 부담 없이 사용할 수 있는 클라우드 파운드리는 두 가지를 들 수 있다. 첫째로, 피보탈 웹 서비스PWS, Pivotal Web Services는 아마존 웹 서비스의 미국 동부에서 동작하며 피보탈이 직접 운영하는 클라우드 파운드리로서 다른 클라우드 서비스와 마찬가지로 가입을 통해 사용할 수 있다. 개발자의 랩탑에 CF라고 불리는 명령행 도구만 준비되면 회원 가입 후 제공되는 계정으로 로그인해서 즉시 사용이 가능하다. 두 번째로, PCFdev로 불리는 환경은 개발자 랩탑 또는 팀에서 사용하는 데스크탑에 설치해서 사용할 수 있는 '개발용' 도구다. 실 서비스 목적으로 사용하는 것은 전혀 권고되지 않으며, 개발팀 로컬에서 빌드된 코드의 빠른 배포를 통한 확인을 지원한다.

두 가지 서비스 모두 '마켓플레이스'를 통해 여러분의 애플리케이션에 직접 바인딩해서 사용할 수 있는 지원 서비스를 제공한다. `cf marketplace` 명령어는 여러분이 준비한 환경에서 사용 가

능한 다양한 서비스를 보여줄 것이다. PWS의 경우에는 다양한 외부 지원 서비스를 사용할 수 있으며, PCFdev의 경우 처음 설치할때 옵션 제공을 통해 MySQL, 래빗엠큐, 레디스, 스프링 클라우드 서비스[SCS]를 함께 사용할 수 있다. 물론 서비스 브로커를 추가해 확장하는 것도 가능하다.

이 외에 여러분이 관심있는 다른 서비스를 사용하고자 하는 경우에는, 아마도 랩탑에 이미 설치되어있을 도커[Docker] 환경을 통해 적절한 도커 이미지를 사용할 수 있을 것이다. 이때 도커를 이용해 준비된 컨테이너의 네트워크 주소 정보 또는 인증 정보를 `cf cups`(`cf create-user-provided-service`의 줄임) 명령어를 써서 클라우드 파운드리에 제공해 다른 서비스 사용과 동일하게 애플리케이션에 바인딩할 수 있다.

이번 부록 B에서는 피보탈 웹 서비스[PWS]와 PCFdev 환경의 준비와 CF 명령행 도구의 사용 방법에 대해 알아보고, 본문에 소개된 예제와 마이크로서비스로 구현된 '클라우드 네이티브 옷집'을 배포하는 방법을 살펴 보겠다.

피보탈 웹 서비스 준비와 사용

피보탈 웹 서비스[Pivotal Web Services]를 사용하려면 우선 가입을 해야 한다. 먼저 피보탈 웹 서비스 사이트(https://run.pivotal.io)에 접속해 보자. 첫 화면에는 클라우드 파운드리에 애플리케이션을 배포하는 방법이나 서비스를 준비하는 방법이 간략하게 소개되어 있다. 가장 크게 보이는 문구는 '무료로 가입해 보세요[Sign up for free]'이며, $87에 상당하는 자원을 무료로 제공한다고 설명되어 있다.

화면 중앙의 초록색 Sign up for free 버튼을 누르면 계정 생성을 위한 정보 입력 화면이 나타난다. 이름과 이메일 주소, 비밀번호 등 간단한 사항을 입력하고 약관을 살펴본 후에 동의 버튼의 체크박스를 확인하고 Sign Up 버튼을 클릭하자. 곧바로, 입력한 이메일 주소로 확인 메일이 발송된다. 이 이메일에 수신된 링크를 클릭해서 메일 계정이 실제로 존재함을 확인하자.

가입이 완료되면, 로그인을 해 보자. 로그인 후에 왼쪽 메뉴에서 Tools를 클릭하면 운영체제에 맞는 CLI 도구를 내려받을 수 있다. 그리고 방금 생성한 계정 정보를 바탕으로 PWS 환경에 로그인하는 명령어가 소개된다.

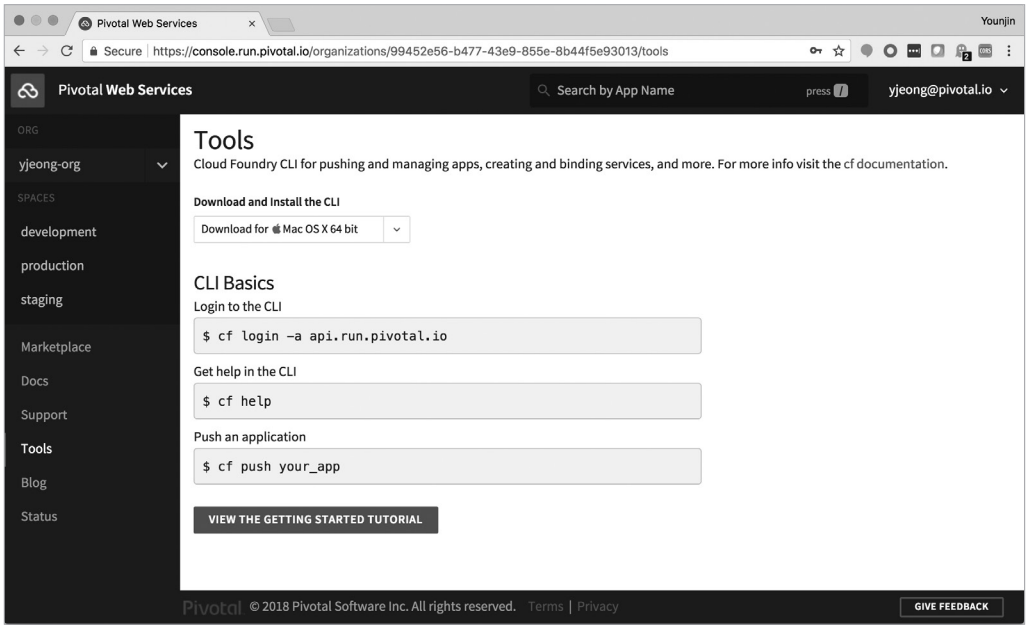

그림 B-1 CF 클라이언트 명령행 도구 다운로드 화면

`cf login` 명령어로 PWS에 로그인을 하면 이제 사용할 준비를 모두 마쳤다. PCFdev가 필요한 것이 아니라면 'cf 명령행 도구 사용' 절로 건너뛰자.

PCFdev 준비와 사용

PCFdev는 랩탑이나 데스크탑에 설치해서 개발용으로 사용이 가능한 PCF의 간소화 버전이다. 한 가지 주의할 점이 있는데, PCFdev는 개발 환경을 지원하기 위한 것이지 실 서비스 환경으로 제공되는 것이 아니므로 실제 서비스를 구동하기 위한 환경으로 사용하면 안 된다. 설치 방법에 대한 내용은 다음 링크에서 확인할 수 있다.

- 자세한 설치 정보: https://docs.pivotal.io/pcf-dev/index.html
- 간소화된 설치 정보: https://pivotal.io/platform/pcf-tutorials/getting-started-with-pivotal-cloud-foundry-dev/introduction

설치 순서는 다음과 같다.

- 사전에 필요한 준비사항
- CF 명령행 도구 설치
- CF 명령행 도구 플러그인 설치
- PCFdev 설치

PCFdev는 현재 다양한 버전으로 제공된다. 이 책의 번역을 진행하는 2018년 5월을 기준으로 가장 최신 버전은 맥 OS X 운영체제용이다. 만약 리눅스나 클라우드 서비스 공급자의 가상 머신을 이용하는 경우에는 별도의 버전을 선택해서 설치하면 된다. PCFdev의 설치에는 피브넷PivNet이라고 불리는 피보탈 네트워크 페이지(https://network.pivotal.io)에서 회원 가입을 해야 한다. 설치 과정에서 이 계정의 ID와 비밀번호를 요구하므로 빠뜨리지 말고 가입하자. 이 피브넷 페이지에서는 피보탈의 다양한 도구를 내려받을 수 있다. 상단의 검색창에 PCF Dev라고 입력하고 PCF Dev 항목에 들어가면 기본으로 제공되는 릴리스와 버전을 확인할 수 있다. 이 버전들은 정규 버전의 PCF와 대응을 표시하는데, 이 부록에서는 리눅스용 v0.30-PCF1.11.0-linux.zip 파일을 사용했다. 이 파일을 내려받아 두자. 만약 PCFdev를 이미 사용해 본 경험이 있거나 맥 OS X 사용자라면 PAS 2.1.0 이상의 버전을 사용할 수 있다.

먼저 버추얼박스VirtualBox 5.0 이상의 버전이 대상 시스템에 설치되어 있어야 한다. 만약 없다면 설치하고자 하는 운영체제의 종류에 따라 버추얼박스 사이트(https://www.virtualbox.org/wiki/Downloads)에서 설치하자. 버추얼박스 이외에도 애플리케이션을 빌드하고 배포하려면 JDK가 설치되어 있어야 한다. 이 책을 읽는 독자분들은 아마 대부분 준비가 되어 있을 것이므로 JDK, JRE 설치 부분은 생략하기로 한다.

다음으로는 6MB 정도의 클라우드 파운드리 명령행 도구가 필요하다. 피보탈 사이트(https://goo.gl/6x1j9F)에서 내려받아 설치하자. 설치가 완료되었다면 `$ cf help` 명령을 실행해서 명령행 도구가 정상적으로 동작하는지 확인한다.

이제 피브넷에서 다운로드한 `cf` 플러그인 파일을 실행한다. 이 도구는 기존 `cf` 명령행 도구에 `cf dev` 명령을 사용할 수 있는 기능을 제공한다. 윈도우와 리눅스에서는 이 파일의 압축을 해제하고 실행하면 된다. 설치가 완료되면 `$ cf dev status` 명령을 실행해서 `cf dev` 플러그인이 제대로 동작하는지 확인해 보자.

이 dev 플러그인의 역할은 1.11 버전까지는 지원되는 윈도우, 리눅스, 맥의 3개 운영체제에서 모두 동일하다. PCFdev의 패키지를 내려받고, 로컬 호스트에 가상 머신을 준비하고, 준비된 가상 머신에 클라우드 파운드리 패키지를 설치한다. 클라우드 파운드리만 설치하는 것이 아니라, 스프링 클라우드 서비스[SCS, Spring Cloud Service], 레디스, MySQL, 래빗엠큐가 함께 제공되어 책에 설명되는 예제와 서비스를 함께 준비해서 사용하기에 용이하다.

하지만 PAS 2.x 버전 이후에는 그 사용 방법이 조금 다르다. 소프트웨어가 지속적으로 업데이트됨에 따라 필요한 구성요소가 늘어났다. 이 버전부터는 보쉬[BOSH]라고 불리는 배포 도구가 로컬 호스트에 함께 설치되어 PCF의 PAS 도구와 좀 더 완벽한 호환성을 제공한다. 이 부록에서 소개되는 내용을 PCFdev를 통해 진행하고자 한다면, 피브넷에서 0.30 버전을 사용하자. 만약 이후의 최신 버전을 사용한다면, PCFdev의 최신 문서를 참고해야 할 필요가 있을 수 있다.

PCFdev는 윈도우, 리눅스, 맥(OS X)을 지원한다. 이 세 가지 운영체제 모두 설치 흐름은 비슷하지만 다운로드 받아야 하는 파일이 다르다. 따라서 가장 최적의 설치 방법은 https://docs.pivotal.io/pcf-dev/index.html의 가이드 또는 PivNet의 해당 패키지 다운로드 페이지의 우측에 설명된 가이드를 따르는 것이 좋다.

표 B-1 PCFdev 설치 옵션(PCFdev 0.30 버전까지 유효함)

플래그(flag)	설명
-c [코어 수]	PCFdev에 할당할 코어의 숫자를 지정
-d [도메인]	PCFdev에 할당할 도메인
-i [IP 주소]	시스템에 있는 IP를 지정하고자 하는 경우
-k	도메인 인증서를 추가하고자 하는 경우
-m [메모리-mb]	MB 단위로 할당하고자 하는 메모리를 입력
-r	SSL 없이 사용하고자 하는 도커(Docker) 레지스트리
-s [서비스]	레디스, 래빗엠큐 등 PCFdev와 함께 준비할 서비스
-t	user / user 계정 활성화

PCFdev는 한번 설치되고 나면 코어 수 조정이나 메모리 크기 조정이 불가능하며, 환경을 지운 후에 다시 만들어야 한다. 따라서 표 B-1에서 설명한 설치 옵션을 바탕으로 편의에 맞게 골라서 사용하면 되겠다.

예제 B-1 PCFdev 환경 설치

```
# 기본 메모리 4GB로 환경 준비
$ cf dev start

# 4코어 8GB, 가능한 모든 서비스 준비
$ cf dev start -c 4 -m 8192 -s all

# IP와 도메인 지정
$ cf dev start -i 192.168.100.100 -d example.io
```

예제 B-1의 명령을 시작하면 PCFdev를 내려받고, 버추얼박스를 사용해서 PCFdev 가상머신을 준비한다. 네트워크 속도에 따라 다르지만 보통 수십 분 정도 걸린다. 설치를 수행 중이라면 기다리는 동안 커피 한 잔 하는 것도 좋은 생각이다. 다음 예제 B-2와 같이 명령을 실행한 후 그림 B-2와 같은 화면이 떴다면 클라우드 파운드리가 모두 준비된 것이다.

예제 B-2 PCFdev 환경 시작

```
$ cf dev start
Downloading Resources...
Starting VPNKit ...
Starting the VM...
Deploying the BOSH Director...
Deploying CF...


To begin using CF Dev, please run:
  cf login -a https://api.v3.pcfdev.io --skip-ssl-validation

Admin user => Email: admin / Password: admin
Regular user => Email: user / Password: pass
```

```
2. @1ccdfb3fd231:/ (bash)
blog @ yjeong $ cf dev start
Downloading VM...
Progress: |++==================>| 100%
VM downloaded.
Allocating 4096 MB out of 16384 MB total system memory (4567 MB free).
Importing VM...
Starting VM...
Provisioning VM...
Waiting for services to start...
7 out of 58 running
7 out of 58 running
7 out of 58 running
7 out of 58 running
47 out of 58 running
58 out of 58 running
 _______  _______  _______    ______   _______  __   __
|       ||       ||       |  |      | |       ||  | |  |
|    _  ||       ||    ___|  |  _    ||    ___||  |_|  |
|   |_| ||       ||   |___   | | |   ||   |___ |       |
|    ___||      _||    ___|  | |_|   ||    ___||       |
|   |    |     |_ |   |      |       ||   |___  |     |
|___|    |_______||___|      |______| |_______|  |___|
is now running.
To begin using PCF Dev, please run:
   cf login -a https://api.local.pcfdev.io --skip-ssl-validation
Apps Manager URL: https://apps.local.pcfdev.io
Admin user => Email: admin / Password: admin
Regular user => Email: user / Password: pass
blog @ yjeong $
```

그림 B-2 PCFdev 0.30 버전 설치 완료 화면

PCFdev는 일반 PCF와 유사한 웹 인터페이스를 제공한다. 도메인을 별도로 지정하지 않았다면 브라우저를 사용해서 https://apps.local.pcfdev.io에 접근해 보자. 인증을 물어오면 아이디 admin, 비밀번호 admin을 사용해서 로그인하면 된다. 이 정보들은 그림 B-2와 같이 설치 완료 후 정보로 표시된다.

이제 클라우드 파운드리를 명령행 도구로 사용해 보자.

CF 명령행 도구 사용

클라우드 파운드리는 cf로 불리는 명령행 도구를 제공한다. cf는 많은 클라우드 도구와 마찬가지로, 클라우드 파운드리가 제공하는 API를 사용하도록 구성된 클라이언트 도구다. 웹 페이지 역시 동일한 API를 사용한다. cf는 사용하기가 매우 간편하며 설치에 필요한 의존성이 그다지 복잡하지 않다. 따라서 애플리케이션 자동화를 위한 스크립트 또는 지속적 전달과 배포Continuous Delivery and Deploy 환경과 쉽게 연동할 수 있다. 테스트 파이프라인 제일 마지막에 붙여 스테이징 환경에 배포를 자동으로 수행하게 하는 경우에 매우 유용하다.

cf 도구를 사용해서 클라우드 파운드리를 사용하려면 예제 B-3과 같이 먼저 로그인을 하자.

예제 B-3 cf를 사용한 클라우드 파운드리 로그인

```
# PWS인 경우
$ cf login -a https://api.run.pivotal.io

# PCFdev인 경우
$ cf login -a https://api.local.pcfdev.io --skip-ssl-validation

# 별도의 도메인을 사용하도록 구성한 경우
$ cf login -a https://api.example.io --skip-ssl-validation
```

클라우드 파운드리가 사용하는 도메인에 도메인 인증서가 적절히 제공되었다면 `--skip-ssl-validation` 파라미터는 필요하지 않다. 준비된 클라우드 파운드리에 애플리케이션을 배포해 보자. 여기서는 스프링 부트 예제로 자주 사용되는 spring-music 프로젝트를 사용해 보겠다(예제 B-4). spring-music 프로젝트는 다음의 링크에서 찾아볼 수 있다.

https://github.com/cloudfoundry-samples/spring-music

예제 B-4 spring-music 프로젝트 다운로드와 빌드

```
# git clone
$ git clone https://github.com/cloudfoundry-samples/spring-music
Cloning into 'spring-music'...
remote: Counting objects: 1197, done.
remote: Total 1197 (delta 0), reused 0 (delta 0), pack-reused 1197
Receiving objects: 100% (1197/1197), 830.97 KiB | 257.00 KiB/s, done.
Resolving deltas: 100% (413/413), done.

# 복제된 디렉토리로 이동
$ cd spring-music

# 빌드
$ ./gradlew build
...

BUILD SUCCESSFUL in 17s
5 actionable tasks: 5 executed
```

이 과정은 여러분들이 애플리케이션을 개발할 때 항상 수행하는 작업일 것이다. 스프링 부트를 사용한다면 스프링 이니셜라이저를 통해 메이븐Maven이나 그레이들Gradle을 선택적으로 사용할 수 있을 것이다. 이렇게 빌드를 통해 준비된 애플리케이션을 클라우드 파운드리에 배포해 보자(예제 B-5).

예제 B-5 spring-music을 클라우드 파운드리에 배포

```
# 클라우드 파운드리 로그인. PWS나 PCFdev 환경에 맞는 로그인을 사용하자
$ cf login -a https://api.run.pivotal.io
API endpoint: https://api.run.pivotal.io

Email> user@email.address
Password>
Authenticating...
OK

API endpoint:   https://api.run.pivotal.io (API version: 2.111.0)
User:           your@email.address
Org:            your-org
Space:          development

# 클라우드 파운드리에 배포. 이 경우 PWS이지만, 방법은 PCFdev 역시 동일하다.
$ cf push
Pushing from manifest to org user-org / space staging as user@email.address...
Using manifest file /tmp/spring-music/manifest.yml
Getting app info...
Creating app with these attributes...
+ name:       spring-music
  path:       /private/tmp/spring-music/build/libs/spring-music-1.0.jar
+ memory:     1G
  routes:
+ spring-music-tired-wombat.cfapps.io

Creating app spring-music...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
Uploading files...
515.25 KiB / 515.25 KiB [==============================================] 100.00% 2s
```

```
Waiting for API to complete processing files...

Staging app and tracing logs...
  Cell 16408489-8b20-458e-9fd6-90ef965c7278 successfully created container for
instance 215030fe-4c2b-4fce-b688-54bd039e784e
  Downloading app package...
  Downloaded app package (40.6M)
  -----> Java Buildpack v4.12 (offline) | https://github.com/cloudfoundry/java-
buildpack.git#5dca820
  -----> Downloading Jvmkill Agent 1.12.0_RELEASE from https://java-buildpack.
cloudfoundry.org/jvmkill/trusty/x86_64/jvmkill-1.12.0_RELEASE.so (found in cache)
  -----> Downloading Open Jdk JRE 1.8.0_172 from https://java-buildpack.cloudfoundry.
org/openjdk/trusty/x86_64/openjdk-1.8.0_172.tar.gz (found in cache)
         Expanding Open Jdk JRE to .java-buildpack/open_jdk_jre (2.0s)
          JVM DNS caching disabled in lieu of BOSH DNS caching
  -----> Downloading Open JDK Like Memory Calculator 3.13.0_RELEASE from https://java-
buildpack.cloudfoundry.org/memory-calculator/trusty/x86_64/memory-calculator-3.13.0_
RELEASE.tar.gz (found in cache)
          Loaded Classes: 19544, Threads: 250
  -----> Downloading Client Certificate Mapper 1.6.0_RELEASE from https://java-
buildpack.cloudfoundry.org/client-certificate-mapper/client-certificate-mapper-1.6.0_
RELEASE.jar (found in cache)
  -----> Downloading Container Security Provider 1.13.0_RELEASE from https://
java-buildpack.cloudfoundry.org/container-security-provider/container-security-
provider-1.13.0_RELEASE.jar (found in cache)
  -----> Downloading Spring Auto Reconfiguration 2.4.0_RELEASE from https://java-
buildpack.cloudfoundry.org/auto-reconfiguration/auto-reconfiguration-2.4.0_RELEASE.jar
(found in cache)
  Exit status 0
  Uploading droplet, build artifacts cache...
  Uploading build artifacts cache...
  Uploading droplet...
  Uploaded build artifacts cache (131B)
  Uploaded droplet (86.5M)
  Uploading complete
  Cell 16408489-8b20-458e-9fd6-90ef965c7278 stopping instance 215030fe-4c2b-4fce-b688-
54bd039e784e
  Cell 16408489-8b20-458e-9fd6-90ef965c7278 destroying container for instance
215030fe-4c2b-4fce-b688-54bd039e784e
  Cell 16408489-8b20-458e-9fd6-90ef965c7278 successfully destroyed container for
instance 215030fe-4c2b-4fce-b688-54bd039e784e
```

```
Waiting for app to start...  ❶

name:              spring-music
requested state:   started
instances:         1/1
usage:             1G x 1 instances
routes:            spring-music-tired-wombat.cfapps.io  ❷
last uploaded:     Sat 19 May 14:59:57 KST 2018
stack:             cflinuxfs2
buildpack:         client-certificate-mapper=1.6.0_RELEASE container-security-
provider=1.13.0_RELEASE java-buildpack=v4.12-offline-https://github.com/cloudfoundry/
java-buildpack.git#5dca820
                   java-main java-opts java-security jvmkill-agent=1.12.0_RELEASE open-
jdk-...
start command:     JAVA_OPTS="-agentpath:$PWD/.java-buildpack/open_jdk_jre/bin/
jvmkill-1.12.0_RELEASE=printHeapHistogram=1 -Djava.io.tmpdir=$TMPDIR
                   -Djava.ext.dirs=$PWD/.java-buildpack/container_security_
provider:$PWD/.java-buildpack/open_jdk_jre/lib/ext -Djava.security.properties=$PWD/.
java-buildpack/java_security/java.security
                   $JAVA_OPTS" && CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_
jre/bin/java-buildpack-memory-calculator-3.13.0_RELEASE -totMemory=$MEMORY_LIMIT
-loadedClasses=20254
                   -poolType=metaspace -stackThreads=250 -vmOptions="$JAVA_OPTS")
&& echo JVM Memory Configuration: $CALCULATED_MEMORY && JAVA_OPTS="$JAVA_OPTS
$CALCULATED_MEMORY" && MALLOC_ARENA_MAX=2
                   SERVER_PORT=$PORT eval exec $PWD/.java-buildpack/open_jdk_jre/bin/
java $JAVA_OPTS -cp $PWD/. org.springframework.boot.loader.JarLauncher

     state     since                  cpu      memory         disk           details
#0   running   2018-05-19T06:01:39Z   131.4%   305.7M of 1G   169.3M of 1G
```

다소 많은 양의 로그 메시지를 볼 수 있다. 자세한 설명은 미뤄두고, 애플리케이션이 정상적으로 배포되어 동작하는지 확인해 보자. 예제 B-5에서 "Waiting for app to start..." 항목 ❶ 바로 아래에 보면, routes: 항목 ❷에 현재 애플리케이션이 동작하는 FQDN[Fully Qualified Domain Name]을 볼 수 있다. 여기에 나타난 주소를 웹 브라우저에 입력해 보자. 이 주소는 {random-word} 설정으로 인해 배포하는 사용자마다 다르게 설정되어 있으므로, 배포 후에 표시되는 routes 항목의 주소를 참조해야 한다. 그림 B-3과 같은 페이지를 볼 수 있다면 성공이다.

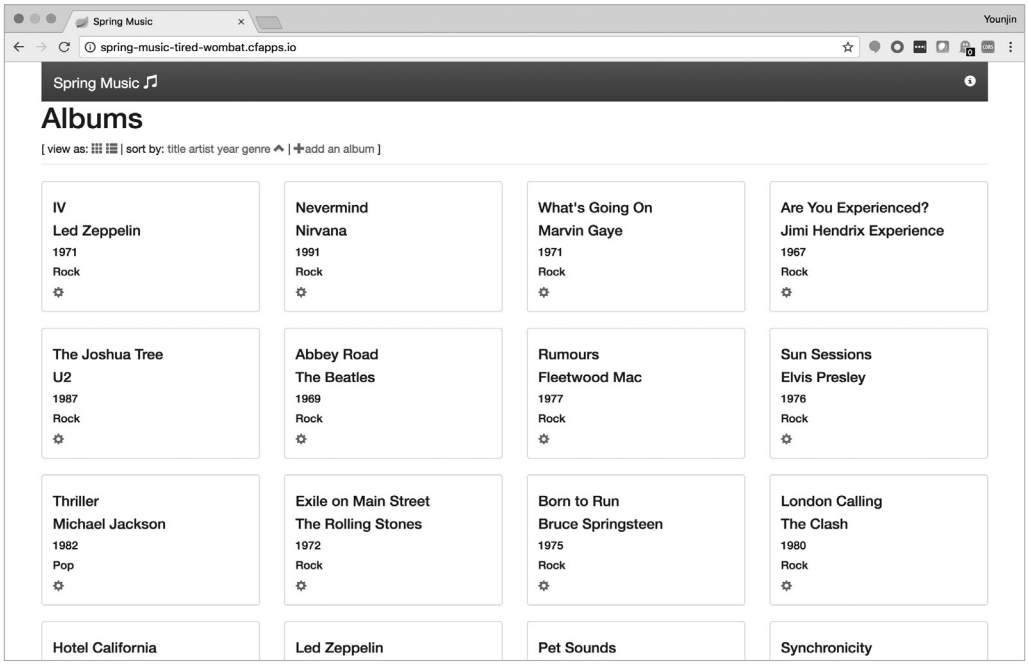

그림 B-3 스프링 뮤직 애플리케이션 웹 페이지

새롭게 배포한 spring-music 애플리케이션은 현재 로드밸런싱이 가능한 상태로 1개의 컨테이너에서 동작 중이다. PWS를 사용하고 있다면 여러분의 애플리케이션은 현재 미국 동부의 아마존 웹 서비스 환경에 배포된 것이다. 애플리케이션을 빌드하고 배포해서 동작을 확인하는 데까지 걸린 시간과, 현재 여러분이 개발해서 운영하는 환경에서의 시간 및 편의성을 비교해서 생각해 보자. 이 마법 같은 동작은 여러분의 애플리케이션이 위치한 디렉토리에서 `cf push` 명령어를 통해 수행되었다. 수많은 로그가 나왔는데, 이 로그의 내용을 대략적으로 설명하면 다음과 같다.

- 지정된 ORG와 SPACE에 애플리케이션을 배포한다는 메시지를 볼 수 있다. ORG와 SPACE는 조직과 조직에서 관리하는 환경을 묘사한다. 예를 들어 검색 엔진을 만드는 팀이라면 SEARCH-ORG, 스페이스는 DEV/STAG/PROD로 나눠 관리할 수 있는 형태로 이해하면 쉽다.
- 디렉토리에서 manifest.yml 파일을 찾았으므로, 이 파일에서 배포 정보를 얻는다. 파일에는 애플리케이션에 사용할 FQDN 정보와 애플리케이션의 이름, 그리고 빌드가 위치한 경로 등이 적혀 있다. manifest.yml은 온라인 상점 배포 부분에서 다시 자세히 다뤄보겠다.

- spring-music 애플리케이션의 논리 공간을 생성하고 경로를 매핑한다. 만약 이때 동일한 라우팅 이름이 존재한다면 에러가 발생한다.
- 지정된 빌드 파일을 업로드한다. 자바의 경우에는 파일이지만, PHP나 노드제이에스 같은 프로젝트들의 경우 디렉토리를 전부 업로드한다.
- 업로드가 완료되면 업로드된 파일이 어떤 언어와 프레임워크로 되어 있는지를 분석해서 적절한 런타임을 적용한다. 런타임의 적용은 '빌드팩buildpack'이라는 패키징된 도구를 사용한다. 이 로그에서는 자바 빌드팩을 적용하고 있다. 만약 여러분이 노드제이에스 애플리케이션을 업로드했다면, package.json 파일의 존재 여부를 확인하고 이를 바탕으로 노드제이에스 빌드팩이 적용될 것이다.
- 적용된 빌드팩을 바탕으로 실행 환경을 구성하는 '스테이징staging' 작업을 시작한다. 스테이징 작업은 여러분의 애플리케이션과 런타임, 그리고 부수적으로 필요한 도구들을 함께 타르볼tarball로 묶어 드롭릿Droplet이라는 파일을 생성하는 과정이다. 이렇게 만들어진 드롭릿은 이후 컨테이너에 배포되어 실행 옵션과 함께 제공되어 애플리케이션을 동작하는 역할을 한다. 즉 이뮤터블 아티팩트Immutable Artifact를 시스템에서 관리한다.
- 스테이징 과정이 완료되면, 스테이징을 수행한 태스크용 컨테이너를 없앤다.
- 새로운 LRPLong Running Process용 컨테이너를 만들어 스테이징으로 생성된 드롭릿을 이 안에 넣고 주어진 환경 변수 등을 주입한 후 드롭릿을 실행한다.
- 애플리케이션이 정상적으로 동작을 시작하면 클라우드 파운드리의 라우터에 매핑된 경로와 컨테이너의 주소를 연결한다. 이후 외부로부터 트래픽을 받을 준비가 된다.

이 같은 일련의 과정은 여러분의 애플리케이션이 서버에 배포될 때마다 항상 반복적으로 수행되어야 하는 동작이다. 또한 이렇게 배포된 애플리케이션은 항상 동작 상태가 모니터링되고, 사용자가 지정한 숫자로 동작하지 않는 경우에는 자동으로 복구를 지원한다. 확장과 축소를 위해서는 예제 B-6과 같이 `cf scale` 명령을 사용할 수 있다.

예제 B-6 클라우드 파운드리의 애플리케이션 확장과 축소

```
$ cf scale spring-music -i 2
OK
$ cf app spring-music
...
```

```
     state      since                  cpu    memory         disk           details
#0   running    2018-05-19T06:01:39Z   0.2%   397.3M of 1G   169.3M of 1G
#1   starting   2018-05-19T06:26:23Z   0.0%   44.9M of 1G    169.3M of 1G
```

애플리케이션은 자동으로 2개의 컨테이너로 확장되고, 정상적으로 구동이 완료되면 로드밸런싱에 자동으로 추가된다. 애플리케이션의 로그를 확인하려면 예제 B-7과 같은 명령어를 사용해 보자.

예제 B-7 클라우드 파운드리에 배포된 애플리케이션의 로그 확인

```
$ cf logs spring-music
# 위의 명령을 수행하고 브라우저에서 새로고침을 수행해 보자

  2018-05-19T15:28:44.13+0900 [RTR/5] OUT spring-music-tired-wombat.cfapps.io - [2018-
05-19T06:28:43.642+0000] "GET /appinfo HTTP/1.1" 200 0 36 "http://spring-music-tired-
wombat.cfapps.io/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36" "10.10.2.26:63624"
"10.10.148.155:61134" x_forwarded_for:"59.10.41.253, 10.10.2.26" x_forwarded_
proto:"http" vcap_request_id:"96e46039-ad46-49ce-57d6-248650f70461" response_
time:0.488234163 app_id:"c6531b8f-c7f7-4bbb-8343-921f4acc9f7f" app_index:"1" x_b3_
traceid:"5e53355785a22d14" x_b3_spanid:"5e53355785a22d14" x_b3_parentspanid:"-"
  2018-05-19T15:28:44.13+0900 [RTR/5] OUT
  2018-05-19T15:28:44.14+0900 [RTR/1] OUT spring-music-tired-wombat.cfapps.io - [2018-
05-19T06:28:43.693+0000] "GET /albums HTTP/1.1" 200 0 4669 "http://spring-music-tired-
wombat.cfapps.io/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36" "10.10.2.26:10131"
"10.10.148.155:61134" x_forwarded_for:"59.10.41.253, 10.10.2.26" x_forwarded_
proto:"http" vcap_request_id:"a4f2c156-74da-47f0-733e-b9d44c06b817" response_
time:0.45598155 app_id:"c6531b8f-c7f7-4bbb-8343-921f4acc9f7f" app_index:"1" x_b3_
traceid:"b63d14720ccec39e" x_b3_spanid:"b63d14720ccec39e" x_b3_parentspanid:"-"
  2018-05-19T15:28:44.14+0900 [RTR/1] OUT
...
```

애플리케이션에서 발생하는 로그는 클라우드 파운드리 내부의 로그리게이터loggregator라는 마이크로서비스로 취합되고, 이렇게 취합된 로그는 애플리케이션 별로 `cf logs [애플리케이션_이름]`의 명령어로 실시간 확인할 수 있다. `cf logs` 명령을 수행하면 클라이언트는 클라우드 파운드리와 웹소켓 연결을 로그 메시지 확인을 위해 구성한다. 만약 실시간 로그가 아니라 최근의 로그를 받아보고 싶은 경우라면 `cf logs [애플리케이션_이름] --recent` 명령어를 사용하면 된다. 로

그 스트림을 중지하려면 Ctrl + C키를 누르면 된다.

현재 이 애플리케이션에는 데이터베이스가 연결되어 있지 않다. 애플리케이션에 데이터베이스를 연결해 보자. spring-music 애플리케이션은 스프링 부트로 만들어져 있으며, MySQL을 사용할 수 있다. 웹 브라우저로 spring-music에 접속해서 오른쪽 상단의 i 버튼을 클릭하면 현재 애플리케이션이 사용하는 설정 프로파일 정보와 연결된 서비스 정보가 나타난다. 아직 데이터베이스를 연결하지 않았으므로 services에는 아무런 정보가 없을 것이다.

PWS에서는 ClearDB라는 서비스를 통해 MySQL을 무료로 사용할 수 있다. PCFdev에서는 p-mysql을 통해 MySQL 서비스 사용이 가능하다. 각 환경에서 MySQL 데이터베이스 종류 확인과 생성을 위해서는 다음과 같은 명령어를 사용하면 된다(예제 B-8).

예제 B-8 클라우드 파운드리 데이터베이스 플랜 확인

```
# PWS
$ cf marketplace -s cleardb
Getting service plan information for service cleardb as ...
OK

service plan   description                                                free or paid
spark          Great for getting started and developing your apps         free
boost          Best for light production or staging your applications     paid
amp            For apps with moderate data requirements                   paid
shock          Designed for apps where you need real MySQL reliability, power and
throughput   paid

# PCFdev

$ cf marketplace -s p-mysql
```

여기 나타나는 서비스 플랜들은 14장의 내용을 참고하면 된다. PWS의 경우 spark 플랜이 무료로 제공되고 있다. 이 플랜을 사용해서 데이터베이스를 생성하고, spring-music 애플리케이션에 연결해 보자. PCFdev의 경우 로컬호스트에서 동작하므로 제공되는 플랜을 사용하면 된다.

예제 B-9 무료 데이터베이스 생성

```
# PWS 데이터베이스 생성
$ cf create-service cleardb spark spring-music-db
OK
```

```
# 생성된 데이터베이스와 spring-music 애플리케이션 바인딩
$ cf bind-service spring-music spring-music-db
OK

# 새로운 설정을 애플리케이션에 반영
$ cf restage spring-music
... (스테이징이 다시 실행됨)
```

스테이징이 완료되고 애플리케이션이 동작한다고 나오면, 브라우저에서 새로고침 버튼을 눌러보자. 오른쪽 상단의 i 버튼을 통해 내용을 보면, spring-music-db가 서비스로 연결되었음을 확인할 수 있을 것이다. 여기서 사용되는 애플리케이션 이름과 서비스 이름은 여러분이 원하는 대로 변경이 가능하다. 그리고 생성한 서비스는 같은 스페이스 안에 동작하는 애플리케이션들이 바인딩해서 사용이 가능하다. 만약 다른 환경에 동작하는 서비스를 사용하고자 한다면 사용자 정의 서비스를 생성해서 애플리케이션에 바인딩하면 된다. 예를 들어, 네이버 서비스를 외부 서비스로 등록하고 이를 애플리케이션에 바인딩하는 것도 가능하다. 예제 B-10을 살펴보자.

예제 B-10 네이버를 사용자 정의 서비스로 생성

```
# uri를 값으로 가지는 사용자 정의 서비스 생성
$ cf cups naver-service -p uri

uri> https://www.naver.com/
OK
```

이와 같이 외부의 데이터베이스나 API를 제공하는 마이크로서비스를 지원 서비스로 명시할 수 있다. 즉 애플리케이션을 MySQL이나 레디스처럼 다른 서비스에 바인딩할 수 있는 서비스로서 등록해서 사용할 수 있다는 것이다. 예를 들어, 스프링 클라우드의 설정 서버Config Server 또는 서비스 디스커버리Service Discovery를 사용하는 경우, 이 애플리케이션을 cf push로 배포하고, 이 애플리케이션의 uri를 사용자 정의 서비스로 등록하면 다른 스프링 부트 애플리케이션에서 이 두 개의 애플리케이션을 서비스로 바인딩해서 사용할 수 있다는 것이다. 이렇게 하면 모든 애플리케이션에 설정 서버나 디스커버리 서비스를 설정하지 않고도 이 기능이 필요한 애플리케이션에서 바인딩해서 사용할 수 있어 사용자가 설정에 신경쓰는 대신 시스템이 알아서 하도록 처리할 수 있다.

이 책의 본문에서 제공되는 다양한 지원 서비스들은 거의 대부분 PWS나 PCFdev에서 사용이 가능하지만, 아파치 카프카의 경우는 아직 지원하고 있지 않다. 이 경우 도커 환경을 통해 카프카 인스턴스를 배포해서 사용할 수 있는데, 이때 PCFdev가 로컬에서 함께 동작한다면 로컬에 도커로 배포된 카프카를 PCFdev에서 사용자 정의 서비스로 등록해서 연결할 수 있겠다.

애플리케이션의 배포를 위해 사용하는 manifest.yml 파일의 구조를 살펴볼 필요가 있다. 이 파일을 이용해 애플리케이션은 매번 동일한 형태로 클라우드 파운드리에 배포될 수 있다. spring-music 프로젝트의 루트 디렉토리를 살펴보면 manifest.yml 파일을 발견할 수 있다.

예제 B-11 spring-music의 manifest.yml 파일

```
---
applications:
- name: spring-music
  memory: 1G
  random-route: true
  path: build/libs/spring-music-1.0.jar
```

spring-music 애플리케이션을 `cf push` 명령으로로 배포하는 경우 manifest.yml 파일이 이미 있다면 이 파일을 사용해서 배포를 수행한다. 만약 없다면, 다음과 같은 명령어 한 줄로 manifest.yml 파일의 내용을 재구성할 수 있다.

```
$ cf push spring-music -m 1024 -p build/libs/spring-music-1.0.jar --random-route
```

이 책 전반에서 사용한 예제들은 깃허브(http://github.com/Cloud-Native-Java)에 공개되어 있는데, 예제들 중 manifest.yml이 포함된 것도 있고 아닌 것들도 있다. 이들은 클라우드 파운드리에 배포해서 사용할 수 있는 예제들이라고 보면 되겠다(단, 한번에 동작하지 않는 경우에는 `cf logs app_name --recent`를 통해 확인해 보자). 만약 여러분이 직접 만든 스프링 부트 애플리케이션을 클라우드 파운드리에 위와 같이 명령행으로 배포를 한 상태라면 manifest.yml 파일이 없을 것이다. 이 경우에는 `cf create-app-manifest [app_name]`으로 manifest.yml을 생성할 수 있다.

예제 B-12 manifest.yml 파일 생성

```
$ cf create-app-manifest spring-music
Creating an app manifest from current settings of app spring-music in org user-org /
space staging as user@email.address...
```

```
OK
Manifest file created successfully at ./spring-music_manifest.yml

$ cat spring-music_manifest.yml
applications:
- name: spring-music
  disk_quota: 1G
  instances: 2
  memory: 1G
  routes:
- route: spring-music-tired-wombat.cfapps.io
  services:
- spring-music-db
- config-server
  stack: cflinuxfs2
```

생성된 manifest.yml을 살펴보면 원래의 파일보다 훨씬 자세한 정보가 들어 있음을 알 수 있다. 기존에 없었는데 생긴 항목은 `services:`와 `stack:`이다. 이중 `services:`는 애플리케이션에 매번 바인딩되어야 하는 서비스를 명시하기 때문에 중요하다. 여기에 사용된 서비스의 이름은 `cf create-service`로 생성할 때 주어진 지원 서비스에 할당한 이름이다. 예제 B-12에서 MySQL을 spring-music-db로 만들어 연결하고 config-server 역시 바인딩했으므로 이 두 개가 manifest.yml에 추가된 것을 볼 수 있다.

PWS 서비스와 PCFdev에서는 서비스 디스커버리나 설정 서버, 서킷 브레이커의 서비스들은 클라우드 파운드리에서 SCS[Spring Cloud Services]로 제공되어 사용할 수 있다. PWS에서 스프링 클라우드 서비스의 영구 사용은 비용을 지불해야 하지만, 학습을 위해 사용하는 경우라면 7일간 무료로 사용이 가능하다.

다른 서비스들과는 달리 스프링 클라우드 설정 서버의 경우 설정 서버가 참조할 원본의 위치를 지정해야 한다. 스프링 클라우드 서비스의 설정 서버는 `git`과 `Vault` 두개의 원본을 지원하는데, `git`을 기준으로 사용하는 방법을 살펴보자.

아마 이 책 본문의 내용을 진행하면서 필요한 내용은 저자 조쉬와 케니가 미리 만들어둔 클라우드 네이티브 자바 깃허브 저장소를 클론해서 사용할 것이다. 여기에 별도로 설정을 위한 깃허브 저장소를 만들었다면, 여기에 여러분의 경로를 사용하면 된다. PWS를 사용하는 경우라면 사설 저장소의 비밀번호 등의 설정 정보 노출에 유의하자.

예제 B-13 스프링 클라우드 서비스의 생성과 사용

```
# PWS 환경에서의 config-server 플랜 확인
$ cf marketplace -s p-config-server
Getting service plan information for service p-config-server as user@email.address...
OK

service plan   description
free or paid
standard       Standard Plan
paid
trial          Service instances using this plan are deleted automatically 7 days
after creation   free

# 무료 플랜을 사용해서 서비스 생성
$ cf create-service -c '{"git": { "uri": "https://github.com/spring-cloud-samples/
config-repo", "repos": { "cook": { "pattern": "cook*", "uri": "https://github.com/
spring-cloud-services-samples/cook-config" } } }, "count": 1 }' p-config-server trial
config-server
Creating service instance config-server in org user-org / space staging as user...
OK

Create in progress. Use 'cf services' or 'cf service config-server' to check operation
status.

# config-service가 정상적으로 생성되었는지 확인

$ cf service config-server
Showing info of service config-server in org user-org / space staging as user@email.
address...

name:            config-server
service:         p-config-server
bound apps:
tags:
plan:            trial
description:     Config Server for Spring Cloud Applications
documentation:
dashboard:       https://spring-cloud-service-broker.cfapps.io/dashboard/p-config-
server/UUID

Showing status of last operation from service config-server...
```

```
status:    create in progress
message:
started:   2018-05-19T07:13:17Z
updated:   2018-05-19T07:18:27Z

# 생성한 config server를 애플리케이션에 연결
$ cf bind-service config-server spring-music
```

만약 PWS를 사용 중이고, 뉴렐릭NewRelic 서비스에 계정이 있다면 손쉽게 뉴렐릭 서비스를 spring-music 서비스와 연동할 수도 있다. 다음 예제 B-14를 살펴보자.

예제 B-14 spring-music 서비스에 뉴렐릭 서비스를 연동.

```
$ cf create-service newrelic standard my-newrelic
Creating service instance my-newrelic in org user-org / space staging as user@email.
address...
OK

$ cf service my-newrelic
Showing info of service my-newrelic in org user-org / space staging as user@email.
address...

name:            my-newrelic
service:         newrelic
bound apps:
tags:
plan:            standard
description:     Manage and monitor your apps
documentation:
dashboard:       https://cloudfoundry.appdirect.com/api/custom/cloudfoundry/v2/sso/
start?serviceUuid=UUID

Showing status of last operation from service my-newrelic...

status:    create succeeded
message:
started:   2018-05-19T07:45:37Z
updated:   2018-05-19T07:45:37Z

$ cf bind-service spring-music my-newrelic
Binding service my-newrelic to app spring-music in org user-org / space staging as
```

```
user@email.address...
OK
TIP: Use 'cf restage spring-music' to ensure your env variable changes take effect
```

브라우저에서 https://run.pivotal.io에 로그인하면, 여러분의 ORG와 SPACE에 spring-music 배포를 찾을 수 있을 것이다. spring-music을 클릭하면 다음과 같은 화면을 볼 수 있다.

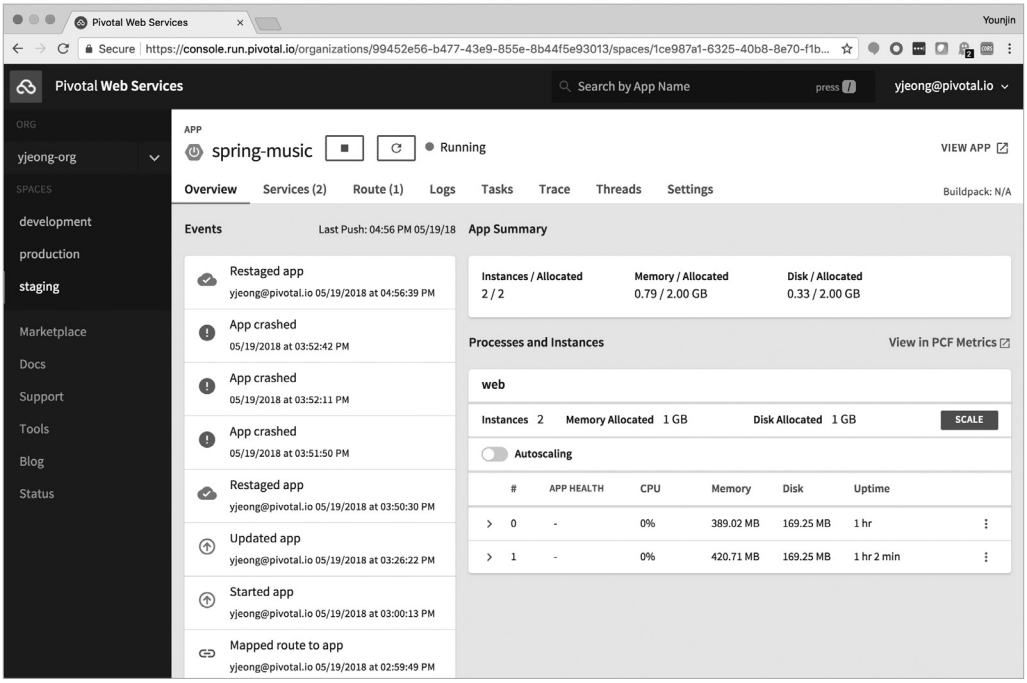

그림 B-4 spring-music을 PWS 웹 콘솔에서 살펴본 모습

그림 B-4의 상단부에서 전원 버튼 모양의 스프링 부트 로고를 볼 수 있는데, 클라우드 파운드리는 이 spring-music 애플리케이션이 스프링 부트를 사용했음을 인지하고 있다. 메뉴 부분에는 다양한 탭이 있는데, 기본 화면인 Overview에서는 현재 애플리케이션의 상태 이벤트와 어떤 상태로 배포되었는지를 보여준다. 두 번째 탭인 Services를 클릭하면 애플리케이션에 바인드된 서비스 목록이 나타난다.

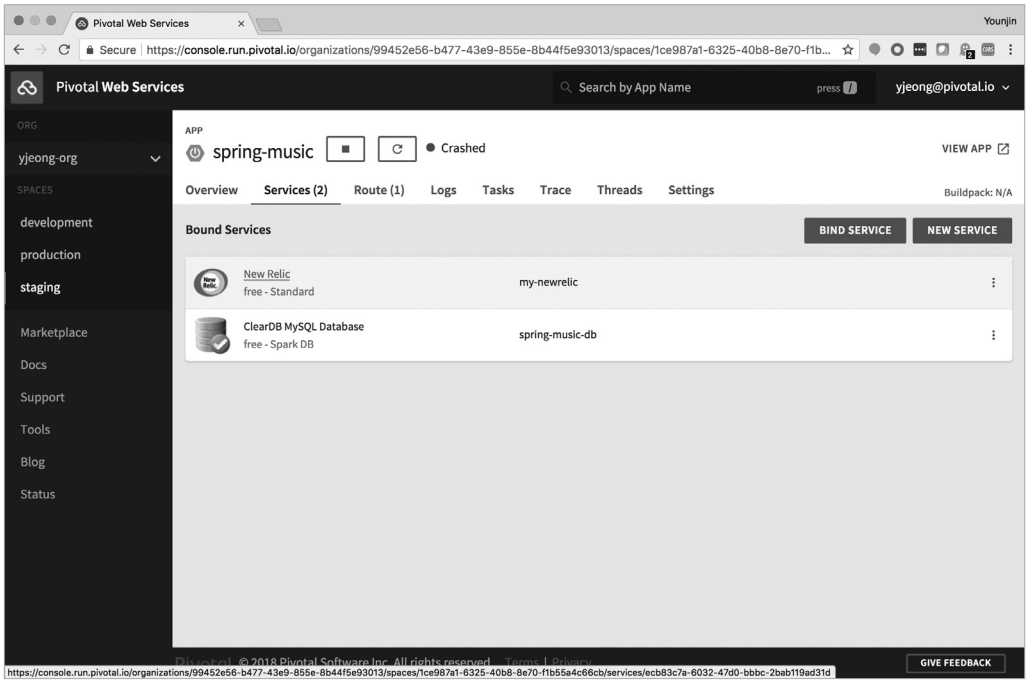

그림 B-5 spring-music에 할당된 서비스를 PWS 웹 콘솔로 확인

그림 B-5의 화면에서 New Relic 서비스 부분을 클릭하면 `Service Key Credential` 토큰을 입력할 수 있는 창이 있다. 여기에 뉴렐릭으로부터 할당받은 라이센스 키를 입력하고, `$ cf restage spring-music` 명령으로 `cf restage`를 실행해본다. `cf restage`는 애플리케이션에서 사용할 시스템의 환경 변수나 새로운 서비스에 대한 정보를 공급하는 역할을 수행한다. 이는 기존에 생성된 컨테이너에서 동작하는 앱에는 새로운 정보가 없기 때문에 수행하는 것으로 이해하면 쉽다. 리스테이징이 완료되면 브라우저에서 spring-music을 새로고침해보자. 그러면 우측 상단의 i에 my-newrelic이 포함되어 있음을 확인할 수 있다. 그리고 여러분의 뉴렐릭 계정으로 접근하면 다음 그림 B-6과 같이 spring-music의 APM 정보가 유입되는 것을 확인할 수 있을 것이다.

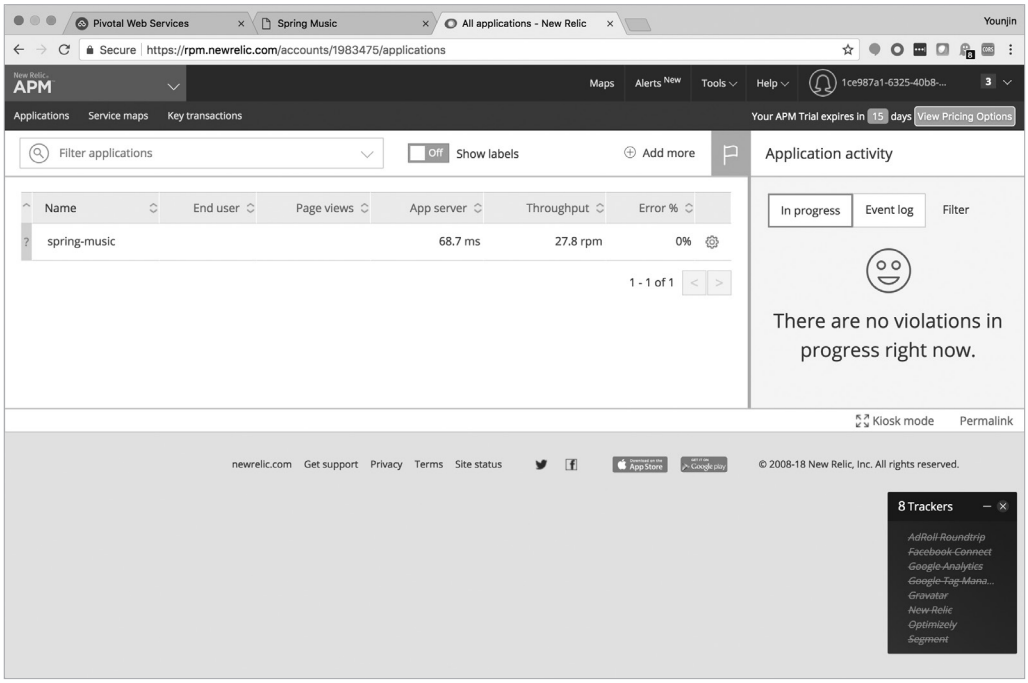

그림 B-6 뉴렐릭 관리 화면. spring-music이 목록에 유입됨

그림 B-7은 뉴렐릭 사용자라면 익숙한 화면일 것이다. 별도의 에이전트 설정이나 설치 과정이 없이, 클라우드 파운드리를 통해 뉴렐릭 서비스를 애플리케이션에 손쉽게 연동한 결과로 그림 B-7과 같은 화면을 볼 수 있다.

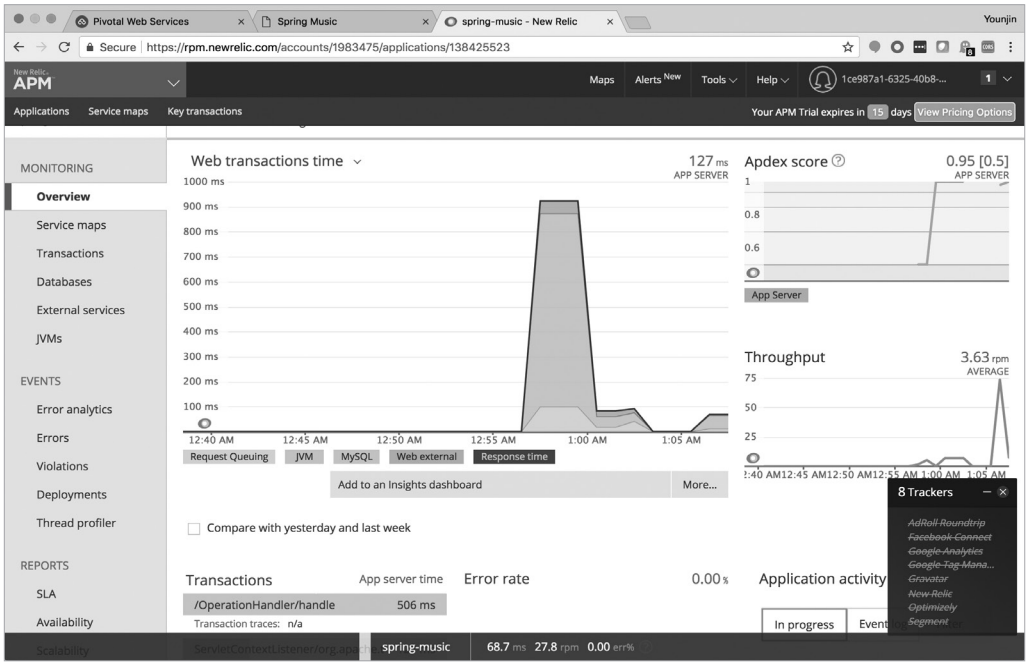

그림 B-7 뉴렐릭 spring-music 애플리케이션 성능 모니터링 대시보드

이런 방식의 서비스 연동은 개발자가 더 이상 복잡한 환경을 직접 설정하지 않아도 됨을 의미한다. 애플리케이션에 필요한 지원 서비스를 클라우드 파운드리에 준비해두면, 개발팀은 언제든 이 서비스들을 가져다가 사용할 수 있는 것이다. 이런 서비스들의 목록은 `cf marketplace` 명령어를 통해 확인이 가능하며, 운영팀에서 만약 개발팀이 필요로 하는 서비스가 없다고 판단되면 https://network.pivotal.io에서 제공하는 도구를 가져다가 준비하는 형태로 제공할 수 있다. 만약 여기에 없는 경우라면 14장에서 소개되는 '클라우드 파운드리 서비스 브로커' 내용을 참조해서 서비스를 공급할 수 있다. 또는 앞에서 한번 소개된 사용자 정의 서비스를 사용하는 방법도 있다. 이와 같은 편의는 비단 애플리케이션 성능 모니터링뿐만 아니라 외부 서비스에 로그를 저장할 때도 사용할 수 있다. 이번에는 spring-music 애플리케이션에서 발생하는 로그를 페이퍼트레일(https://papertrailapp.com/) 서비스로 보내서 분석이나 검색 등에 활용해 보자.

먼저 페이퍼트레일 서비스에서 계정을 생성하자. 계정 생성이 완료되면 로그를 전달할 엔드포인트를 할당해 준다. 찾을 수 없다면 로그인 후 우측 상단에서 Settings ❯ Log Destinations를 참조해 보면 `logx.papertrailapp.com:` 포트의 형태로 된 주소를 찾을 수 있을 것이다. 이 정보를 사용해서 다음 예제 B-15와 같이 사용자 정의 서비스를 생성해준다.

예제 B-15 페이퍼트레일 로그 서비스 생성과 spring-music 서비스와 연결

```
$ cf cups my-logs -l syslog-tls://logsN.papertrailapp.com:XXXXX
OK

$ cf bind-service spring-music my-logs
OK

$ cf restart spring-music
...
```

이제 페이퍼트레일의 이벤트 대시보드를 열어 둔 상태에서 spring-music 앱이 열린 브라우저를 새로고침하면 애플리케이션의 로그가 페이퍼트레일로 유입되는 것을 확인할 수 있다(그림 B-8). 유입된 로그에서 이벤트를 검색하거나 에러나 기타 다양한 방법으로 애플리케이션이 동작하며 남기는 정보를 확인할 수 있다. 이렇게 로그를 다른 서비스로 보내는 것은 스모로직SumoLogic이나 스플렁크Splunk 또는 ELK와 같은 다양한 서비스로 처리할 수 있다. 애플리케이션에서 시스템에 로그를 남길 수 있는 방법이 무엇이든 사용하기만 하면 나머지는 클라우드 파운드리가 처리한다.

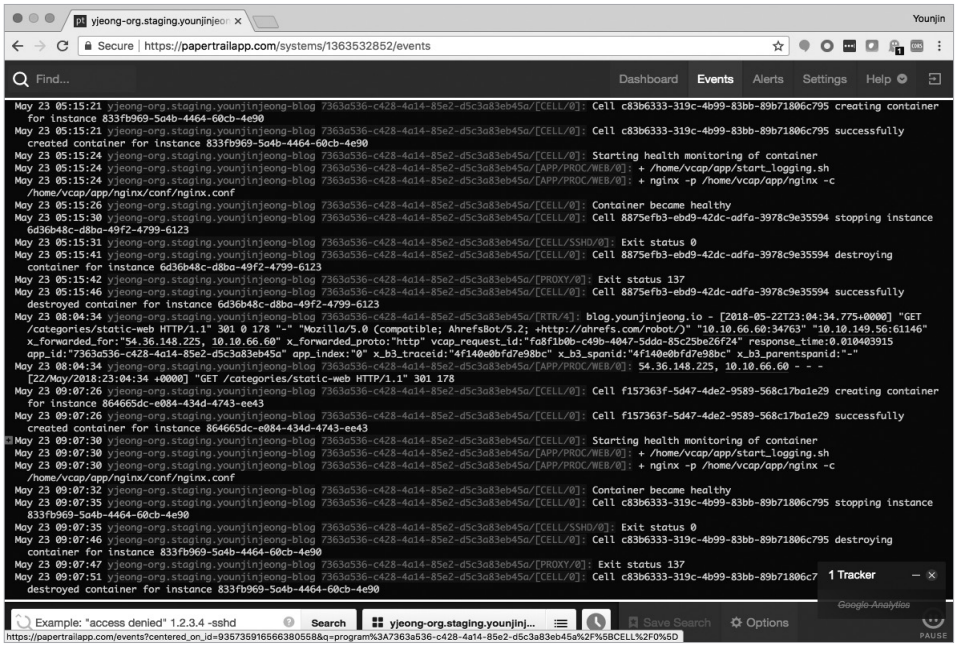

그림 B-8 페이퍼트레일 로그 유입 화면

이 외에도 클라우드 파운드리는 애플리케이션의 로그와 성능 정보 등을 한눈에 볼 수 있는 메트릭을 제공하기도 한다. PWS에 로그인해서 이전처럼 spring-music의 세부 정보 화면으로 들어가면, Overview 탭에서 View in PCF Metrics 링크를 찾을 수 있다. 이 링크를 클릭하면 여러분이 배포한 spring-music 애플리케이션이 동작하는 컨테이너의 다양한 상태 정보와 로그를 확인할 수 있다. 여기에는 애플리케이션의 평균 응답 지연시간, 분당 유입된 요청 수, 메모리 사용 등과 함께 로그 이벤트를 확인할 수 있다. 따라서 애플리케이션 배포 후에 어떤 문제가 발생했다면, 이런 화면을 통해 한눈에 확인할 수 있는 것이다.

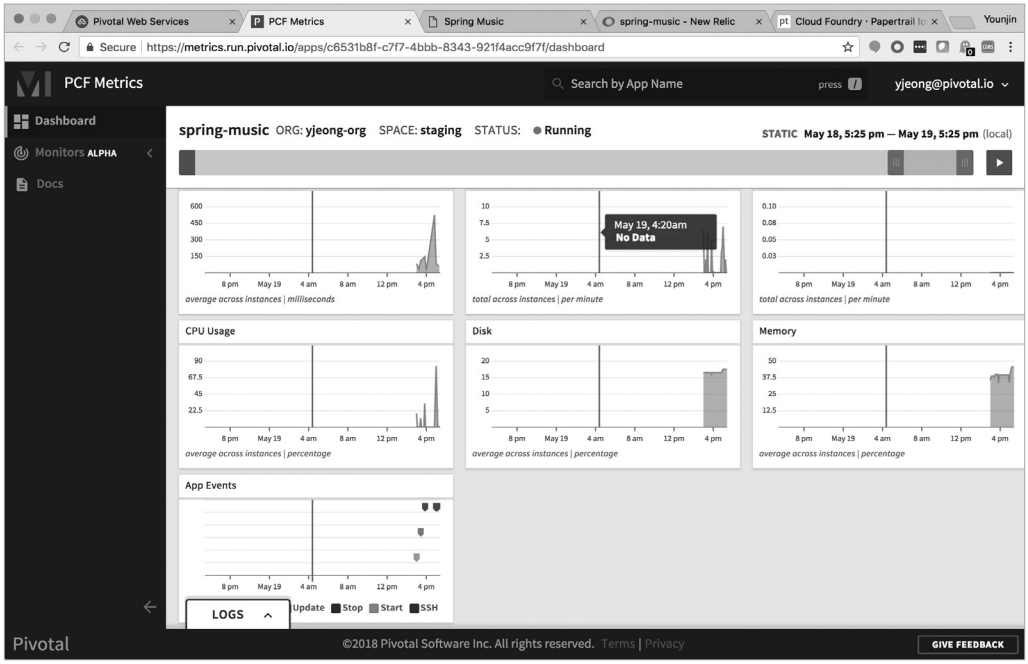

그림 B-9 PCF Metrics 화면

애플리케이션의 배포를 쉽게 하고, 필요한 서비스를 언제든 만들어 바인딩할 수 있는 능력은 매우 중요하다. 플랫폼의 주요 역할은 바로 여기에 있으며, 단순히 컨테이너를 운영하는 것과는 별개로 다운타임을 낮추고, 애플리케이션의 성능 모니터링과 로그를 통합 운영할 수 있는 등의 12요소에서 요구하는 조건을 충족해야 하는 것이다. 클라우드 파운드리는 이런 동작을 금융 및 제조 등의 미션 크리티컬한 산업 분야에서 사용되는 높은 신뢰성을 제공하는 플랫폼이다. 그리고 이런 플랫폼의 확보는 이 책에서 설명하는 다양한 마이크로서비스들이 지속적으로 생성되고 배포되고 업데이트되는 수명 주기를 지원할 것이다.

PWS를 사용한 클라우드 네이티브 자바 예제 코드 배포해 보기

앞서 애플리케이션 간의 메시징에 대해 다룬 10장에서는 래빗엠큐를 사용하며 프로듀서Producer와 컨슈머Consumer의 애플리케이션을 로컬에서 동작하는 방식으로 예제를 제공한다. 하지만 이 예제의 정상적인 구동을 위해서는 스프링 애플리케이션 실행을 위한 준비 외에도 래빗엠큐 인스턴스를 준비해야 동작을 확인할 수 있을 것이다. 이 책에서는 MySQL, 래빗엠큐, 레디스, 몽고디비 등 다양한 서비스 인스턴스를 사용하는데, 이 도구들을 매번 설치해서 준비하는 것은 여간 귀찮은 일이 아니다.

물론 도커Docker 환경을 자유자재로 사용하는 분들에게는 이 서비스들의 준비가 어렵게 느껴지지 않을 수도 있다. 도커와 쿠버네티스를 사용할 수도 있겠지만, 이것은 아마 로컬 랩탑에서 구동하기엔 아무래도 무리일 수도 있겠다.

이런 경우에 PWS 환경은 아주 유용하다. 언급된 거의 대부분의 서비스를 제공하며, 여러분의 스프링 부트 애플리케이션을 구동하는 데 최적의 환경을 제공하기 때문이다. 먼저, 메시징과 관련된 예제 코드를 가져와서 예제 B-16과 같이 몇 가지를 수정 후 빌드하자.

예제 B-16 메시징 예제의 복제와 클라우드 파운드리 동작을 위한 수정

```
$ git clone https://github.com/cloud-native-java/messaging
$ cd messaging

# 컨슈머 예제의 래빗엠큐 인스턴스 참조 주소를 변경
$ vi stream/stream-consumer/src/main/resources/application.properties

# 다음 두 개 라인을 주석 처리
# server.port=0
# spring.rabbitmq.addresses=localhost

# 프로듀서 예제의 래빗엠큐 인스턴스 참조 주소를 변경
$ vi stream/stream-producer/src/main/resources/application.properties

# 다음 1개 라인을 주석 처리
# spring.rabbitmq.addresses=localhost

# 빌드 messaging 프로젝트 루트 디렉토리에서 실행
$ mvn clean package
...
[INFO] Reactor Summary:
```

```
[INFO]
[INFO] messaging ..................................... SUCCESS [  1.308 s]
[INFO] messaging/stream ............................... SUCCESS [  0.004 s]
[INFO] messaging/stream-producer ...................... SUCCESS [  1.826 s]
[INFO] messaging/stream-consumer ...................... SUCCESS [  0.345 s]
[INFO] messaging/batch-and-integration ................ SUCCESS [  0.570 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.537 s
[INFO] Finished at: 2018-05-19T18:56:51+09:00
[INFO] Final Memory: 52M/444M
[INFO] ------------------------------------------------------------------------
```

클라우드 파운드리로의 배포를 위해 application.properties 파일에서 로컬 호스트로 지정된 부분을 주석 처리 했지만, 원한다면 프로파일을 사용하는 방법으로 설정을 재구성할 수도 있다. 또는 스프링 클라우드 설정 서버를 사용하는 방법을 적용할 수도 있다. 여기에서는 파일을 직접 수정했다. 빌드가 완료되었다면 애플리케이션을 배포할 준비가 다 된 것이다. 하지만 먼저 필요한 서비스를 준비해 보자. PWS에서는 래빗엠큐 서비스를 ClearAMQP라는 이름으로 제공한다. 마켓플레이스에서 무료로 제공하는 구성이 무엇인지 확인하고, 이 래빗엠큐 지원 서비스를 준비해 보자(예제 B-17).

예제 B-17 래빗엠큐 서비스의 준비

```
$ cf marketplace -s cloudamqp
Getting service plan information for service cloudamqp as yjeong@pivotal.io...
OK

service plan   description                                free or paid
lemur          Shared cluster with low limits for free    free
tiger          Shared cluster for production apps         paid
bunny          Dedicated server                           paid
rabbit         Dedicated HA cluster                       paid
panda          Dedicated HA cluster with high throughput  paid

# lemur 서비스 플랜만이 무료이므로, 이것을 사용해서 서비스를 생성한다

$ cf create-service cloudamqp lemur my-broker
OK
```

이제 애플리케이션에 사용할 래빗엠큐가 준비되었다. 이제 예제 B-18과 같이 프로듀서와 컨슈머 스프링 부트 애플리케이션을 배포해 보자.

예제 B-18 10장의 메시징 컨슈머와 프로듀서 애플리케이션을 PWS에 배포

```
$ cd stream
$ pwd
messaging/stream

$ cd stream-consumer
$ cf push stream-consumer -p target/stream-consumer.jar --random-route
...
# stream-consumer 앱이 정상적으로 배포되는지 확인한다.

$ cd ../stream-producer/
$ cf push stream-producer -p target/stream-producer.jar --random-route
...
# stream-producer 앱이 정상적으로 배포되는지 확인한다.
```

stream-consumer와 stream-producer의 두 개 앱이 모두 정상적으로 배포되었다면, 이제 10장에 설명된 대로 예제가 정상적으로 동작하는지 확인해 보자. 앱의 정상 동작 확인을 위해서는 먼저 컨슈머 앱의 로그를 `cf logs`로 확인한 상태에서 프로듀서에 `curl`로 `GET` 요청을 수행해 보면 된다. 터미널을 두 개 열거나, 탭을 지원한다면 두 개의 탭에서 실행해 보자.

예제 B-19 애플리케이션 정상 동작 확인

```
# 터미널 1에서
$ cf logs stream-consumer-${random-route}
...

# 터미널 2에서
$ $ curl https://yjeong-producer.cfapps.io/hi/World
Hello, World!

$ curl https://strem-producer-${random-route}.cfapps.io/hi/Again
Hello, Again!
```

각각의 앱이 동작하는 주소는 cf apps 명령을 통해 살펴볼 수 있을 것이다. 애플리케이션이 메시지 큐를 통해 올바르게 데이터를 주고받고 있다면, 다음 예제 B-20과 같은 컨슈머 애플리케이션 로그를 확인할 수 있을 것이다.

예제 B-20 컨슈머 애플리케이션 로그

```
2018-05-19T19:09:34.85+0900 [APP/PROC/WEB/0] OUT 2018-05-19 10:09:34.859  INFO 17 ---
[ngs-p2p-group-1] onsumer$$EnhancerBySpringCGLIB$$5f3ccaf4 : greeting received in
IntegrationFlow (directed): DIRECT: HELLO, WORLD!
  2018-05-19T19:09:34.86+0900 [APP/PROC/WEB/0] OUT 2018-05-19 10:09:34.869  INFO 17
--- [NGerGOr-4DfLQ-1] onsumer$$EnhancerBySpringCGLIB$$5f3ccaf4 : greeting received in
IntegrationFlow (broadcast): BROADCAST: HELLO, WORLD!
  2018-05-19T19:09:45.46+0900 [APP/PROC/WEB/0] OUT 2018-05-19 10:09:45.462  INFO 17
--- [ngs-p2p-group-1] onsumer$$EnhancerBySpringCGLIB$$5f3ccaf4 : greeting received in
IntegrationFlow (directed): DIRECT: HELLO, AGAIN!
  2018-05-19T19:09:45.49+0900 [APP/PROC/WEB/0] OUT 2018-05-19 10:09:45.495  INFO 17
--- [NGerGOr-4DfLQ-1] onsumer$$EnhancerBySpringCGLIB$$5f3ccaf4 : greeting received in
IntegrationFlow (broadcast): BROADCAST: HELLO, AGAIN!
```

동일한 방법으로 9장에서 데이터소스를 필요로 하는 다양한 애플리케이션에 PWS의 마켓플레이스에서 제공하는 서비스를 생성하고, 바인딩해서 동작을 구현해 볼 수 있을 것이다. 물론 이전에 설명한 바와 같이 로컬 호스트에서 도커와 PCFdev를 함께 사용 중이라면 cf cups의 사용자 지정 서비스 생성을 통해 연동하는 것도 가능하다.

클라우드 파운드리로 생성한 환경의 제거

이 책에 소개된 내용을 클라우드 파운드리를 이용해 직접 실행해본 후에 더 이상 유지하고 싶지 않은 경우라면 다음 두 가지를 유의하면 된다.

- PWS를 사용하는 경우라면 각 스페이스에서 동작하는 애플리케이션과 서비스를 적절히 제거하자.
- PCFdev의 경우, cf dev terminate 명령어로 모두 한꺼번에 제거할 수 있다.

PWS의 경우 기본적으로 60일간의 사용을 무료로 제공하지만 60일이 지나면 청구될 가능성이 있다. 또한 무료로 사용할 수 있는 계정에서 신용카드의 등록이 없다면 유료 서비스를 사용하거나, 처음 주어진 용량 이상으로 애플리케이션을 배포하는 것이 불가능하기 때문에 하나의 예제에 대한 동작을 확인했다면 그때그때 지워주는 것이 편리하겠다. 또 레디스와 같은 인스턴스는 PWS에서 계정당 1개의 인스턴스 생성만 허용하는 경우도 있으므로, 생성과 삭제에 주의하자.

애플리케이션이 만약 서비스와 바인딩되어 있다면, 애플리케이션을 지우기 위해서는 바인딩을 해제해야 한다. 이전에 살펴본 spring-music 애플리케이션을 바탕으로 지원 서비스와 바인딩, 그리고 애플리케이션을 환경에서 삭제하는 순서를 알아보자(예제 B-21).

예제 B-21 클라우드 파운드리에 배포된 애플리케이션의 제거

```
# 스페이스에 생성한 서비스의 목록 확인
$ cf services

# 특정 서비스의 세부 사항 확인
$ cf service my-newrelic

# 스페이스에 생성한 애플리케이션의 목록 확인
$ cf apps

# 특정 앱의 세부 사항 확인
$ cf app spring-music

# 애플리케이션과 지원 서비스의 언바인딩
$ cf unbind-service spring-music my-newrelic
$ cf unbind-service spring-music spring-music-db

# 서비스의 삭제
$ cf unbind-service spring-music spring-music-db
Unbinding app spring-music from service spring-music-db in org user-org / space
staging as user@email.address...
OK
spring-music @ user $ cf delete-service spring-music-db

Really delete the service spring-music-db?> yes
Deleting service spring-music-db in org user-org / space staging as user@email.
address...

# 애플리케이션의 삭제
```

```
$ cf delete spring-music

Really delete the app spring-music?> yes
Deleting app spring-music in org user-org / space staging as user@email.address...
OK

# 라우팅의 확인
$ cf routes

# PWS에서 라우팅의 제거
$ cf delete-route cfapps.io --hostname spring-music-tired-wombat

Really delete the route spring-music-tired-wombat.cfapps.io?> yes
Deleting route spring-music-tired-wombat.cfapps.io...
OK
```

삭제 명령은 모두 사용자의 확인을 요한다. 개발이나 스테이징 환경이 아니라 실 서비스 환경이라면 삭제할 때 매우 주의해야 한다. 따라서 스페이스를 사용자에게 할당할 때는 보통 개발환경만을 제공하고, 스테이징과 실 서비스는 CI 및 CD 도구에만 권한을 할당하는 것이 올바른 방법이다. PCFdev는 설치형 환경이므로, 다음과 같이 이 설치 전체를 삭제할 수도 있다.

예제 B-22 PCFdev 환경의 삭제

```
# 버전에 따라
$ cf dev destroy

$ cf dev stop
```

cf dev 환경이 사용하는 캐시나 설정 등의 내용은 보통 [HOME_DIR]/.cfdev에 저장되어 있다. 이후 필요가 없다면 rm 명령어를 사용해서 삭제해 주면 된다. rm 명령어를 사용하는 경우에는 주의하자.

마이크로서비스로 구현된 클라우드 네이티브 옷집 배포해 보기

이 책의 저자 중 한 명인 케니 바스타니Kenny Bastani는 책의 내용을 반영한 예제를 깃허브에 공개했다. 아래의 두 가지 링크에 접속해 보자.

- https://github.com/kbastani/spring-cloud-event-sourcing-example
- https://github.com/kbastani/spring-cloud-event-sourcing-config

위의 링크에 소개된 README 파일을 보면, 5개의 마이크로서비스와 이 마이크로서비스 지원을 위한 스프링 클라우드 서비스로 구성된 옷집 애플리케이션의 구성을 확인할 수 있다. 하나의 프로젝트로 스프링 부트로 만들어진 소스 코드를 포함하고 있으므로, 이 책에서 소개된 다양한 기술들의 응용으로서 살펴보면 도움이 되리라 믿는다.

각 마이크로서비스는 자신만의 데이터소스를 사용하고 있다.

- MySQL: 관계형 데이터베이스
- 네오포제이Neo4J: 그래프 데이터베이스
- 몽고디비MongoDB: 도큐먼트 저장소
- 레디스Redis: 키-값 저장소

이들 중 나머지 세 가지는 PWS 환경의 마켓플레이스에서 모두 제공하지만, 그래프 데이터베이스인 네오포제이는 현재 PWS 마켓플레이스에서 지원하지 않는다. 따라서 재고 서비스의 동작을 위해 필요한 네오포제이 사용을 위해서는 다음과 같은 방법을 사용할 수 있다.

- 아마존이나 다른 가상 머신 환경을 제공하는 클라우드 서비스에 네오포제이를 설치해서 연결하는 방법
- 네오포제이를 제공하는 클라우드 서비스를 사용하는 방법. 시험 버전을 사용하면 보통 무료 사용 옵션을 제공한다.

전자의 경우 네오포제이를 직접 설치해야 하는 어려움이 있을 수 있다. 또한 PWS는 퍼블릭 환경이므로, 이 환경에서 접근하려면 공인 IP 주소 또는 도메인이 필요하다. 이 방법을 사용하면 아마 PWS에서 MySQL이나 레디스를 준비해서 사용하는 것이 얼마나 편리한 방법인지 느껴 볼

수 있을 것이다. 후자의 경우는 네오포제이를 SaaS 모델로 제공하는 서비스를 사용하는 것이다. 역시 계정을 생성하는 불편함은 있지만, 별도의 리소스를 관리하지 않아도 되는 점이 매우 매력적이다. 네오포제이의 공식 페이지에서는 그래프스토리GraphStory(http://www.graphstory.com)와 그래프그리드GraphGrid(http://www.graphgrid.com)의 서비스를 소개하고 있다. 어떤 모델을 사용하건, `cf cups` 명령어를 사용해서 서비스로 등록하면 된다.

스프링 클라우드 서비스

스프링 클라우드 서비스는 여러분이 매번 스프링 이니셜라이저를 통해 설정 서버나 유레카를 매번 만들어 직접 관리하지 않아도 되는 PWS 및 PCFdev에서 제공하는 서비스다. 따라서 이 책 후반부의 서비스 디스커버리나 설정 서버를 사용하는 예제를 쓸 때 스프링 클라우드 서비스를 대신 사용하면 여러모로 수월할 수 있다. 스프링 클라우드 서비스는 PWS 환경에서는 7일간 사용이 무료로 지원된다. 이 서비스를 사용해서 유레카와 설정 서버, 히스트릭스 대시보드를 스프링 클라우드 애플리케이션으로 직접 배포하는 대신 사용할 수 있다. 이 부록에서 소개하는 케니의 블로그와 깃허브의 이벤트 소싱 예제에서는 스프링 클라우드 넷플릭스 프로젝트를 직접 사용하는 방법을 취했다. 소스 디렉토리를 살피면 다음과 같은 스프링 클라우드 넷플릭스 프로젝트들이 있을 것이다.

- config-service(설정 서비스)
- discovery-service(스프링 클라우드 넷플릭스 유레카)
- edge-service(스프링 클라우드 넷플릭스 주울과 리본)
- hystrix-dashboard(스프링 클라우드 넷플릭스 히스트릭스 대시보드)

다른 마이크로서비스들을 배포하기 전, 위 서비스들을 먼저 설정해서 배포해야 한다. PWS나 PCFdev에서 제공하는 스프링 클라우드 서비스에 대한 사용 방법은 다음 링크를 참고하자.

- PWS 스프링 클라우드 서비스: https://docs.run.pivotal.io/spring-cloud-services/
- PCFdev 스프링 클라우드 서비스: https://docs.pivotal.io/pcf-dev/usage-scs.html

설정과 배포 방법 및 과정

다수의 컴포넌트로 이루어진 옷집 서비스를 여러분만의 서비스로 배포하기 위해서 다음과 같은 진행 과정을 살펴보자.

1. 먼저 케니의 깃허브에 있는 코드들을 여러분의 깃허브 계정으로 복사fork한다. 깃허브가 생소하다면, 케니의 event source example 페이지 우측 상단의 Fork 버튼을 누르면 된다.
2. 하나는 소스고, 다른 하나는 이 소스로 빌드되는 애플리케이션에 필요한 설정이다. 이 설정들은 설정 서버에 의해 제공되므로, 여러분의 깃허브에 복제된 config 저장소의 링크를 스프링 클라우드 설정 서버 애플리케이션에 있는 application.properties에 `spring.cloud.config.server.git.uri` 설정을 통해 제공하자.
3. 모든 소스를 빌드한다. 각 하위의 target 디렉토리에 jar 파일이 생성될 것이다.
4. `cf push` 명령어를 통해 설정 서버를 배포하면, 이 서버가 배포된 주소가 제공될 것이다. 이 설정 서버의 주소를 `cf cups` 명령어를 통해 `config-serive`로 만든다.
5. 서비스 디스커버리 역시 마찬가지다. 빌드를 하고 `cf push`를 통해 디스커버리 서비스를 배포하면 나오는 주소를 `cf cups` 명령어를 통해 `discovery-service`로 등록한다.
6. 각 마이크로서비스, 예를 들어 카탈로그 서비스에 보면 manifest.yml 파일이 존재한다. 여기에 보면 `services:` 항목에 이미 생성되어 있어야 할 서비스 목록이 있다. 다른 마이크로서비스에 '서비스'로서 등록되어야 하는 컴포넌트 먼저 배포를 수행하고, `cf cups` 명령어를 사용해서 지정된 이름으로 등록하자.
7. 이런 방법으로 각 서비스를 빌드해서 `cf push` 명령어로 배포를 수행하면 된다.
8. PWS에 배포하는 경우 배포하는 서비스의 라우팅은 전체 PWS에서 유일해야 한다. 따라서 `config-service`와 같이 중복될 가능성이 높은 이름으로 그대로 배포하기보다는, `config-service-${random}` 또는 `cf push config-service --random-route`를 사용해서 중복 발생으로 인한 에러를 피하자.

예제 B-23 설정 서비스의 배포와 등록

```
# 소스코드 디렉토리에서
$ cd config-service
$ vi src/main/resources/bootstrap.yml
```

```
# 제일 하단의 uri: https://github.com/kbastani/spring-cloud-event-sourcing-config.git 부
분을 fork된 여러분의 깃허브 config 주소로 지정한다.

$ ./mvnw clean package
$ cf push   ## config-service의 manifest.yml에 이미 random-route가 정의되어 있어 그냥 사용하면
된다.
...
..
routes:            config-service-{your-name}.cfapps.io
...

$ cf cups config-service -p uri
uri> http://config-service-{your-name}.cfapps.io

OK

$ cf services
...
$ cf service config-service
Showing info of service config-service in org your-org / space your-space as your@
email.address...

name:         config-service
service:      user-provided
bound apps:
```

예제 B-23은 설정 서비스를 빌드하고, cf push로 배포해서 config-service로 등록하는 과정을 보여준다. 그리고 cf service 명령어를 통해 서비스가 정상적으로 등록되었는지 확인한다. 그다음에는 discovery-service를 동일한 방법으로 배포해야 한다. discovery-serivce의 경우 설정을 직접 application.properties 또는 bootstrap.properties에 지정해도 되고, 경우에 따라 설정 서버로부터 설정을 연동해서 사용하기도 한다. 이 케니의 예제 소스에서는 연동하지 않으므로, 그냥 배포하자.

예제 B-24 디스커버리 서비스의 빌드와 등록

```
$ cd discovery-service

$ ./mvnw clean package
...
```

```
$ cf push
...
routes:            discovery-service-{your-name}.cfapps.io
..

$ cf cups discovery-service -p uri
uri> http://discovery-service-{your-name}.cfapps.io

OK

$ cf service discovery-service
Showing info of service discovery-service in org your-org / space your-space as your@
email.address...

name:         discovery-service
service:      user-provided
bound apps:
```

uri를 제공할때, http://를 붙이는 것을 잊지 말자. 예제 B-23과 예제 B-24를 통해 config-service와 discovery-serivce가 준비되었다. 이제 이를 사용하는 마이크로서비스를 배포해 보자. user-service는 사용자의 인증을 처리하는 서비스다. 이 역시 다른 서비스에서 사용되어야 하고, user-service 역시 config-service와 discovery-service에 의존하고 있으며, 자체적으로 MySQL 데이터베이스가 필요하기도 하다. user-service의 배포를 위한 manifest.yml 파일을 살펴보자(예제 B-25).

예제 B-25 user-service의 배포를 위한 manifest.yml 파일

```
---
applications:
  ❶
- name: user-service
  memory: 1024M
  instances: 1
  ❷
  host: user-service-${random-word}
  path: target/user-service-0.0.1-SNAPSHOT.jar
  env:
  ❸
    SPRING_PROFILES_ACTIVE: cloud
```

```
    DEBUG: "true"
    debug: "true"
  ❹
  services:
   - discovery-service
   - config-service
   - user-db
```

❶ name은 클라우드 파운드리 내에서 이 애플리케이션을 식별할 수 있는 이름을 제공한다.

❷ host는 이 애플리케이션을 외부에서 접근하기 위한 라우팅 이름, 즉 FQDN에 사용할 도메인 접두어를 제공한다. 여기에는 user-service-${random-word}를 사용해서 중복을 피하고 있다.

❸ SPRING_PROFILES_ACTIVE를 통해 애플리케이션 설정에 사용할 프로파일을 환경 변수에 지정하고 있다. 여기서는 cloud를 사용하도록 제공하는데, 이 cloud 관련 설정은 여러분이 복제한 깃허브의 설정 저장소에 user-service에 정의되어 있다.

❹ user-serivce가 동작하기 위해 사전에 준비되어야 할 서비스 목록이다. config-service와 discovery-service는 준비되었지만 user-db는 현재 존재하지 않는다.

예제 B-26에서는 user-service에 필요한 MySQL 데이터베이스를 생성하고, user-service를 배포해 보자.

예제 B-26 user-service MySQL 데이터베이스 준비와 user-service 배포

```
$ cd user-service
$ ./mvnw clean package
...
...

# PWS라면, cleardb가 MySQL로 제공되고 있으며, spark가 무료로 사용할 수 있는 규모다.
$ cf create-service cleardb spark user-db
...
...

# PCFdev 라면, p-mysql의 100mb를 사용하면 된다. cs는 create-service의 약자로 사용할 수 있다.
$ cf cs p-mysql 100mb user-db
...
...

$ cf service user-db
Showing info of service user-db in org your-org / space your-space as your@email.
```

```
address...

name:            user-db
service:         p-mysql
bound apps:      user-service
tags:
plan:            100mb
description:     MySQL databases on demand
documentation:
dashboard:       https://p-mysql/[USER-UNIQUE-MANAGEMENT-DASHBOARD-LINK]

$ cf push
...
...
Showing health and status for app user-service in org your-org / space your-space as
your@email.address...

name:              user-service
requested state:   started
instances:         1/1
usage:             1G x 1 instances
routes:            user-service-{your-name}.cfapps.io
last uploaded:     Wed 28 Feb 13:35:52 KST 2018
stack:             cflinuxfs2
buildpack:         java_buildpack_offline

     state     since                  cpu    memory         disk           details
#0   running   2018-02-28T04:37:26Z   0.4%   636.6M of 1G   194.2M of 1G

# user-service도 다른 서비스에 바인딩되어야 한다. 따라서 cf cups 명령어로 서비스로 등록하자.

$ cf cups user-service -p uri
uri> http://user-service-{your-name}.cfapps.io
OK
```

cf create-service 명령어는 마켓플레이스에 이미 등록된 MySQL을 사용할 수 있도록 지원한다. 개발자인 여러분은 이 데이터베이스가 어디서 동작하는지 알 필요 없이, user-service를 배포하기만 하면 된다. 그러면 cf 명령어는 manifest.yml 파일의 내용을 살펴, config-service, discovery-service, user-db를 애플리케이션에 자동으로 바인딩한다.

지금까지의 내용을 바탕으로 하나씩 서비스를 cf push 명령어를 통해 배포할 수 있다. 배포 전에 먼저 manifest.yml 파일의 내용을 살피고, 필요한 서비스가 이미 준비되어 있는지 살핀 후, cf push 명령어를 사용하면 된다. 만약 해당 서비스가 다른 서비스에 바인딩되어야 한다면, cf cups 명령어를 사용해서 등록한다. 그리고 각 서비스에 데이터베이스가 필요하다면, cf create-service 명령어를 사용해서 manifest.yml 파일에 명시된 이름으로 준비한다. 준비해야 할 데이터베이스의 종류와 서비스는 소스 코드와 함께 있는 README를 살펴보자.

네오포제이의 경우 외부 서비스를 여러분이 직접 준비해서 사용해야 한다. 준비가 되었다면 사용자 정의 서비스를 만들어 사용한다. 네오포제이 데이터베이스를 필요로 하는 서비스는 재고 서비스이므로, 재고 서비스의 manifest.yml에 명시된 이름으로 준비된 네오포제이 서비스를 생성해 보자(예제 B-27). 네오포제이 서비스의 준비는 이전에 설명한 세 가지 방법 중 하나, 즉 공인 아이피 주소를 사용하여 PWS 환경에 접근 가능한 서버에 네오포제이를 직접 설치하거나, 그래프스토리GraphStory(http://www.graphstory.com)와 그래프그리드GraphGrid(http://www.graphgrid.com)와 같이 네오포제이 인스턴스를 제공하는 서비스를 사용하는 방법이다. 만약 PCFdev를 로컬 환경에서 사용 중이라면, 도커 환경을 이용해 네오포제이를 준비해서 사용할 수도 있다.

예제 B-27 사용자 지정 네오포제이 서비스 생성

```
$ cf cups inventory-db -p uri
uri> jdbc:neo4j://YOUR_NEO4J_DATABASE_IP:PORT
OK

$ cf service inventory-db
Showing info of service inventory-db in org your-org / space your-space as your@email.
address...

name:         inventory-db
service:      user-provided
bound apps:
```

모든 서비스를 문제 없이 배포했다면, 예제 B-28과 같이 cf apps 명령어를 사용해서 배포된 서비스를 모두 확인해 보자.

예제 B-28 배포된 모든 서비스 확인

```
$ cf apps
Getting apps in org your-org / space your-space as your@email.address...
OK

name                    requested state   instances   memory   disk   urls
account-service         started           2/2         850M     1G     account-service-
{your-name}.cfapps.io
catalog-service         started           2/2         850M     1G     catalog-service-
{your-name}.cfapps.io
config-service          started           1/1         850M     1G     config-service-
{your-name}.cfapps.io
discovery-service       started           1/1         850M     1G     discovery-
service-{your-name}.cfapps.io
edge-service            started           1/1         850M     1G     edge-service-
{your-name}.cfapps.io
hystrix-dashboard       started           1/1         1G       1G     hystrix-
dashboard-{your-name}.cfapps.io
inventory-service       started           1/1         850M     1G     inventory-
service-{your-name}.cfapps.io
online-store-web        started           2/2         850M     1G     online-store-
web-{your-name}.cfapps.io
order-service           started           1/1         850M     1G     order-service-
{your-name}.cfapps.io
shopping-cart-service   started           1/1         1G       1G     shopping-cart-
service-{your-name}.cfapps.io
user-service            started           1/1         1G       1G     user-service-
{your-name}.cfapps.io
```

discovery-service의 url로 접근하면 넷플릭스 유레카 서비스에 각 서비스가 등록된 모습을 확인할 수 있을 것이다. 그리고 여러분의 online-store-web 서비스의 url로 접근하면 그림 B-11과 같은 화면을 볼 수 있다. 만약 페이지에 접근했는데 ERROR: parallax.js requires jQuery의 에러가 발생한다면, 그림 B-10의 화면을 참고해서 크롬 브라우저의 우측 상단에 다음의 버튼을 눌러 모든 스크립트를 불러오기 버튼을 누르면 그림 B-11과 같이 정상적인 첫 화면을 볼 수 있을 것이다.

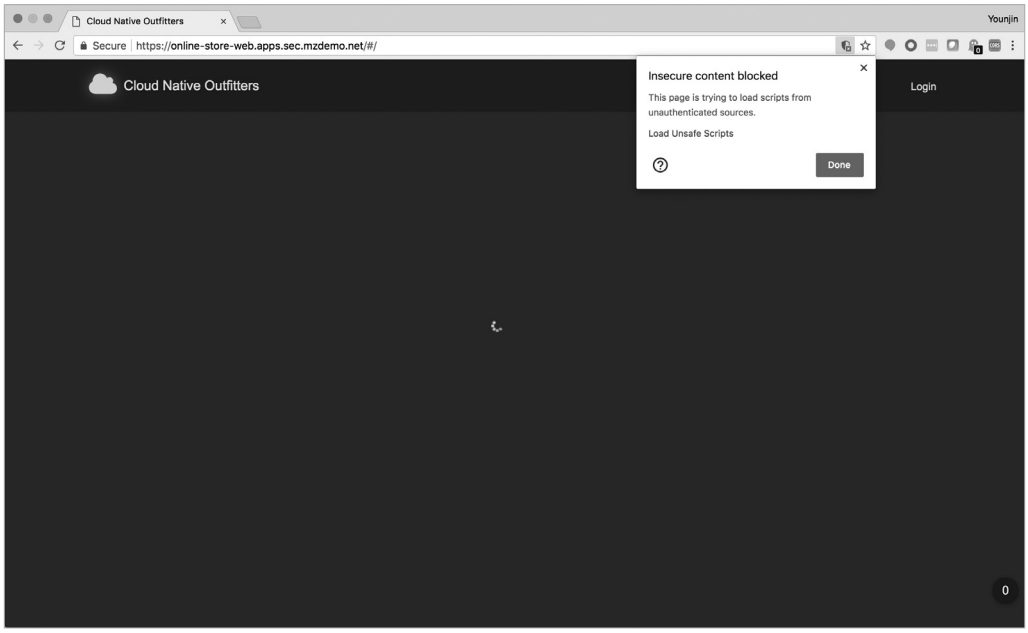

그림 B-10 크로스 사이트 스크립팅 에러 처리

정상적인 페이지는 그림 B-11과 같다. 티셔츠를 판매하는 페이지이며, 로그인 아이디와 비밀번호는 user, password이다. 로그인했을 때와 하지 않았을 때의 페이지 표시 내용이 다르다. 그리고 주문 등을 처리하는 방법이 이벤트 소싱의 방법이며, 각 서비스의 구획이 9장에 소개된 내용을 바탕으로 이루어져 있으므로 참고할 만하겠다.

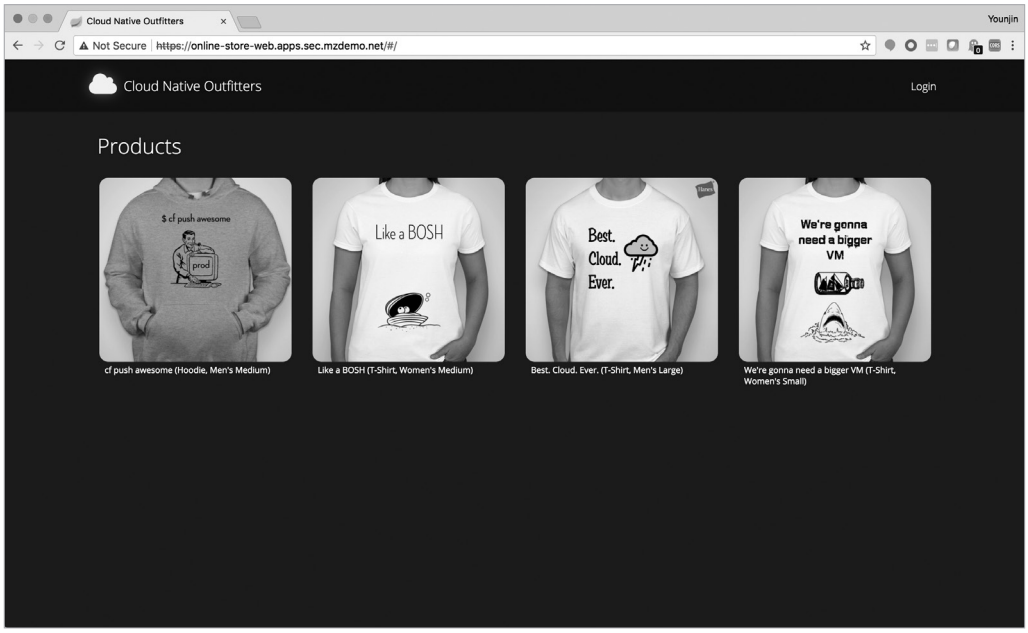

그림 B-11 마이크로서비스로 이루어진 클라우드 티셔츠 판매 페이지

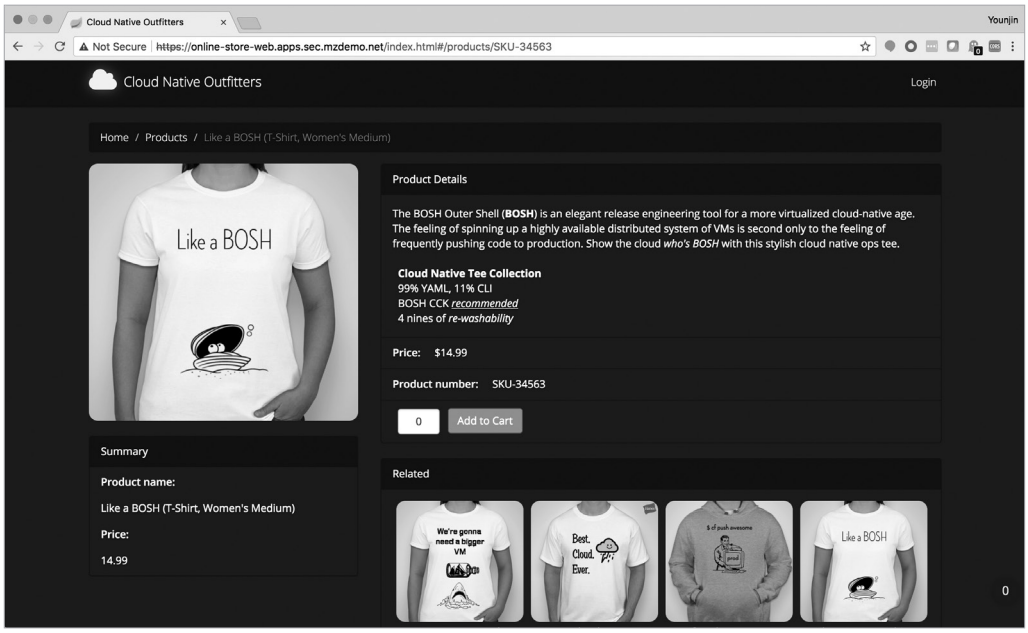

그림 B-12 티셔츠 판매 세부 페이지

이 전체 과정에 여러분이 시간을 얼마나 쏟았는지는 모르지만, 각 서비스별로 서버를 설정하고, 데이터베이스를 설치해서 애플리케이션에 연결하고, 다시 로드밸런싱을 처리하며, 각 애플리케이션의 로그를 원격으로 수집해서 처리할 수 있는데 필요한 시간보다는 훨씬 적게 걸렸을 것이라고 생각한다. 게다가 여러분은 필요하다면 앞에서 살펴본 뉴렐릭과 같은 서비스를 연결해볼 수도 있으며, 로그를 페이퍼트레일 서비스에 전달할 수도 있다. 이와 같이 각각의 마이크로서비스를 개발하는 데는 스프링 부트와 스프링 클라우드 서비스를 사용할 수 있지만, 지원 서비스의 준비와 애플리케이션의 연결, 그리고 클라우드를 기반으로 한 확장과 배포된 애플리케이션의 상시 동작은 클라우드 파운드리의 도움을 받을 수 있다. 즉 클라우드 파운드리의 도움을 받아 개발자나 운영자가 직접 배포에 필요한 반복 작업을 매번 하지 않아도 된다.

정리

클라우드 파운드리는 클라우드에서 애플리케이션의 배포와 운영, 그리고 다양한 서비스의 연동을 통해 개발팀에 강력한 편의를 제공한다. 개발팀에 제공된 강력한 편의는 다시 전체 마이크로서비스 운용을 위해 필요한 로깅이나 메트릭, 텔레메트리 정보 등을 통해 서비스의 운영 상태를 손쉽게 파악하는 동시에 다양한 서비스를 제공함으로써 ITIL과 같은 종래의 티켓을 통한 방식에서 API를 통한 방식으로 운영 자체를 전환한다. 즉 데브옵스로 일의 방법을 전환하는 데 필수적인 개발팀과 운영팀의 계약 관계를 API로 바꾸는 핵심을 제공하는 것이다.

개발팀이 더욱 개발에 집중하고 클라우드에 애플리케이션을 연속적으로 배포할 수 있을 때 마이크로서비스 구조가 발전할 수 있을 것이다. 이것을 위한 생태계가 클라우드 파운드리이며, 운영팀에 의해 개발팀에 제공된다. 넷플릭스의 사례 발표에 따르면 서너 명으로 구성된 운영 조직이 수천 명이 개발하는 거대한 마이크로서비스에 사용되는 핵심 도구를 제공한다고 한다. 주울Zuul 프록시의 경우에도 4명, 이브이캐시EVCache의 경우도 3명 정도다. 마이크로서비스가 선형적으로 전체 서비스 내부에서 증가하더라도, 이를 지원하는 운영팀이 역시 동일하게 증가해야 한다면 그것은 아마도 굉장한 위험일 것이다. 이는 클라우드를 사용하는 것만으로는 해결되지 않는 문제이기도 한다. 클라우드 파운드리는 이 지점에서 수만 명의 개발자가 수만 개의 컨테이너를 사용한 서비스를 운영하더라도 단 여섯 명이 관리 가능한 능력을 제공한다.

지금까지 부록 B에서는 책에 나온 예제를 손쉽게 배포하는 방법을 설명하고, 저자들이 책에서 중간중간 설명한 클라우드 파운드리의 장점을 한꺼번에 살펴봤다. 클라우드 파운드리는 매우 빠르게 발전하고 있는 프로젝트이므로, 만약 부록의 내용이 잘 동작하지 않는다면(버전에 특화된 문제를 제외하고 아예 동작하는 경우는 아마 없을 테지만) 각 항목의 주제를 바탕으로 구글 검색을 통해 피보탈 문서를 살펴보면 되겠다.