Cloud Native Application 배포하기

이 정 구
itgenius1004@gmail.com

Agenda

클라우드 네이티브 소개

개발 환경 구성 및 설정

클라우드 네이티브 패턴 및 기술

스프링 부트 컨테이너화

스프링 부트를 위한 쿠버네티스 기초

클라우드 네이티브 소개

- 클라우드 네이티브가 무엇인가?
- 클라우드와 클라우드 컴퓨팅 모델
- 클라우드 네이티브 애플리케이션의 특징
- 클라우드 네이티브를 지원하는 문화 및 관행
- 클라우드가 최선의 선택인가?
- 클라우드 네이티브 구성
- 클라우드 네이티브 애플리케이션을 위한 아키텍처

클라우드 네이티브 소개

클라우드 네이티브(Cloud Native)는 애플리케이션과 서비스를 효율적으로 개발, 배포 및 운영하기 위해 클라우드 환경을 최대한 활용하는 접근 방식입니다. 확장성, 유연성, 회복력을 강조합니다.

- 컨테이너화(Containerization)
- 마이크로서비스 아키텍처(Microservice Architecture)
- 데브옵스(DevOps)
- 지속적인 통합 및 지속적인 배포(CI/CD)
- 인프라 자동화(Infrastructure Automation)

기업이 빠르게 변화하는 시장 요구사항에 유연하게 대응하고 비용 효율적으로 IT인프라를 운영할 수 있도록 지원합니다.

클라우드 네이티브 소개

클라우드 네이티브란 무엇인가?

폴 프리맨틀(Paul Fremantle)은 2010년 5월 25일에 '클라우드 네이티브'에 대한 블로그 포스트를 작성했는데, 이는 클라우드 네이티브라는 용어를 사용한 최소의 사례임

https://wso2.com/library/articles/2010/05/blog-post-cloud-native/

Distributed/dynamically wired, Elastic, Multi-tenant, Self-service, Granualrly metered & billed, Incrementally deployed & tested

당시 마이크로서비스, 도커, 데브옵스, 쿠버네티스, 스프링 부트와 같은 개념과 기술들이 아직 존재하지 않았을 때, 그는 WSO2 팀과 함께 클라우드 환경에서 잘 작동하는 애플리케이션과 미들웨어가 무엇인지, 즉 클라우드 네이티브가 무엇인지에 대해 논의함

프리맨틀이 설명한 핵심 개념은 클라우드 네이티브 애플리케이션은 클라우드 환경과 클라우드 컴퓨팅 모델의 이점을 활용하도록 특별히 설계되어야 한다는 것임

기존 애플리케이션을 클라우드로 이전하는 '리프트 앤 시프트' 방식은 애플리케이션을 클라우드에 맞게 태생적으로 변화시키지는 않음

클라우드 네이티브란 무엇인가?

- 클라우드 네이티브에서의 세 가지 P
  - What does it mean for applications to be designed specifically for the cloud? The Cloud Native Computing Foundation (CNCF) answers that question in its cloud native definition
    - Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.
    - These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

클라우드 네이티브란 무엇인가?

- 클라우드 네이티브의 세 가지 P 요소
  - 플랫폼(Platforms)
    - 클라우드(공용, 사설, 하이브리드)같은 동적 분산 환경을 기반으로 하는 플랫폼에서 실행
  - 특성(Properties)
    - 확장 가능하고 느슨하게 결합되며 복원력이 뛰어나고 관리가 용이하고 관찰 가능
  - 실행(Practices)
    - 경고한 자동화를 통해 빈번하고 예측 가능한 방식으로 시스템을 변경하는 것이 가능하고 자동화 지속적 전달, DevOps등

클라우드와 클라우드 컴퓨팅 모델

클라우드 네이티브 애플리케이션이 실행되는 환경은 클라우드임

클라우드는 다양한 컴퓨팅 모델을 특징으로 하는 IT 인프라이며 소비자가 필요로 하는 제어 수준에 따라 제공 업체에 의해 서비스로 제공된다.

클라우드와 클라우드 컴퓨팅 모델

클라우드 컴퓨팅 모델이란?

- 미국 국립표준기술연구소(National Institute of Standards and Technology:NIST)
  - Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

클라우드와 클라우드 컴퓨팅 모델

클라우드 배포 모델

- 사설 클라우드
  - 한 단체나 조직에서만 사용하도록 제공하는 클라우드 인프라
- 공공 클라우드
  - 일반 사용자들에게 공공으로 제공되는 클라우드 인프라
- 하이브리드 클라우드
  - 사설 클라우드와 공공 클라우드의 두가지 유형에 묶어서 마치 하나의 단일 환경인 것처럼 제공되는 인프라

클라우드와 클라우드 컴퓨팅 모델

클라우드 컴퓨팅 서비스 모델. 제공하는 추상화 수준, 주체가 어느 수준(플랫폼 또는 소비자)에서 관리할 책임이 있는지에 따라 달라짐.

클라우드와 클라우드 컴퓨팅 모델

- 서비스형 인프라스트럭처(Infrastructure as a service. IaaS)
  - AWS Elastic Compute Cloud EC2, Azure Virtual Machine, Google Compute Engine, Alibaba Virtual Machine, DigitalOcean Droplet
- 서비스형 컨테이너(Container as a service, Caas)
  - Docker Swarm, Apache Mesos, Kubernetes
  - Amazon Elastic Kubernetes Service EKS, Azure Kubernetes Service AKS, Google Kubernetes Engine GKE, Alibaba Container Service for Kubernetes ACK, DigitalOcean Kubernetes
- 서비스형 플랫폼(Platform as a service, PaaS)
  - Cloud Foundry, Heroku, AWS Elastic Beanstalk, Azure App Service, Google App Engine

클라우드와 클라우드 컴퓨팅 모델

- 서비스형 함수(Function as a service, FaaS)
  - Amazon AWS Lambda, Azure functions, Google Cloud Function, Alibaba Function Compute, Knative, Apache OpenWhisk
- 서비스형 소프트웨어(Software as a service, Saas)
  - Microsoft Office 365, Proton Mail, GitHub, Plausible Analytics,

클라우드 네이티브 애플리케이션의 특징

클라우드 네이티브 애플리케이션은 CNCF에 의해 다섯 가지 주요 특성을 가져야 한다고 정의됨.

클라우드는 Where, 클라우드 네이티브는 How

확장성, 느슨한 결합, 복원력, 관찰 가능성, 그리고 관리 가능성. 클라우드 네이티브는 이러한 특성을 나타내는 애플리케이션을 구축하고 운영하는 방법론임.

코넬리아 데이비스(Cornelia Davis)는 “클라우드 네이티브 소프트웨어는 어디에서 컴퓨팅을 하는지가 아닌, 어떻게 컴퓨팅을 하는지 로 정의 된다”고 요약하여, 클라우드 네이티브가 애플리케이션을 어떻게 구축하고 관리하는지에 초점을 맞춘다는 것을 강조함.

클라우드 네이티브 애플리케이션의 특징

클라우드 네이티브 애플리케이션의 특징

- 확장성
  - 수직적 확장(Vertical scalability)
    - 컴퓨팅 노드에 CPU나 메모리 같은 하드웨어 리소스를 추가하거나 제거해서 확장 혹은 축소
  - 수평적 확장(Horizontal scalability)
    - 더 많은 컴퓨팅 노드나 컨테이너를 추가하는 것. 수직적 확장처럼 제약 없음

클라우드 네이티브 애플리케이션의 특징

- 느슨한 결합(Loose Coupling)
  - 결합(Coupling) vs 응집력(Cohension)
  - 모듈화(Modularization) – 파르나스(Parnas)
    - 관리(Managerial)
    - 제품 유연성(Production Flexibility)
    - 이해력(Comprehensibility)

클라우드 네이티브 애플리케이션의 특징

- 복원력(Resilience) – 정상 작동에 대한 오류 및 문제에 직면하여 허용 가능한 수준의 서비스를 제공하고 유지할 수 있도록 하는 하드웨어-소프트웨어 네트워크의 기능
  - 결함(Fault)
  - 오류(Error)
  - 실패(Failure)

클라우드 네이티브 애플리케이션의 특징

- 관측 가능성(Observability)
  - 제어 이론에서 비롯된 속성
  - 애플리케이션의 외부 출력에서 내부 상태를 추론하는 것
  - 관리 용이성은 내부 상태와 외부 상태의 출력을 변경하는 것에 관한 것
  - 두 경우 모두 애플리케이션 아티팩트는 변경되지 않으면 불가면적임

클라우드 네이티브 애플리케이션의 특징

- 모니터링(Monitoring)
- 경고/시각화(Alerting/Visualization)
- 인프라를 추적하는 분산 시스템(Distributed System Tracing Infrastructure)
- 로그 집계/분석(Log Aggregation/Analytics)

클라우드 네이티브 애플리케이션의 특징

- 관리 용이성(Manageability)
  - 제어 이론에서 관측 가능성에 대응하는 개념으로 제어 가능성이 있는데, 외부 입력을 통해 유한한 시간 이내에 시스템의 상태 또는 출력을 변경하는 능력을 의미함
  - Spring Cloud Config Server, ConfigMap & Secret, Kustomize

클라우드 네이티브를 지원하는 문화 및 관행

- 자동화(Automation)
  - 코드형 인프라스트럭처(Infrastructure as code)
  - 코드형 설정(Configuration as code)
  - 테라폼(Terraform)
- 지속적 전달(Continuous Delivery)
  - 지속적 통합(Continuous Integration, CI)
    - 기본 브랜치에 지속적으로 적어도 하루에 한번 커밋, 커밋이 있으면 소스는 자동으로 컴파일되고 테스트되며 실행 가능한 아티팩트(예: JAR 및 컨테이너 이미지)로 패키징됨
    - 오류가 감지되면 즉시 수정함으로써 기본 브랜치는 개발 작업을 문제없이 계속 진행함

클라우드 네이티브를 지원하는 문화 및 관행

- 지속적 전달(Continuous Delivery, CD)
  - CI를 기반으로 하며 기본 브랜치가 문제없이 배포 가능한 상태를 유지하도록 하는데 중점을 둠
  - 변경된 소스가 기본 브랜치로 병합된 결과물로 발생한 아티팩트는 실행 가능한 상태가 됨
  - 소프트웨어는 실제 서비스와 환경이 유사한 환경에 배포됨
  - 배포 유효성(Releasability)
- 배포 파이프라인(Deployment Pipeline)
- 지속적 전달(Continuous Deployment, CD)
  - 배포 파이프라인에 마지막 단계를 하나 추가하여 변경된 사항을 프로덕션에 자동으로 배포함

클라우드 네이티브를 지원하는 문화 및 관행

- 데브옵스(DevOps)
  - 데브옵스는 노옵스를 의미하지 않는다.
  - 데브옵스는 도구가 아니다.
  - 데브옵스는 자동화가 아니다.
  - 데브옵스는 역할이 아니다.
  - 데브옵스는 팀이 아니다.

클라우드가 최선의 선택인가?

클라우드 네이티브로 옮겨갈 때 속도, 복원력, 확장성, 비용 최적화를 달성하려는 목표임

클라우드 네이티브의 목표인 속도, 복원력, 확장성, 비용 최적화 있지 않다면 고려가 필요함

클라우드가 최선의 선택인가?

- 속도
  - 소프트웨어를 더 신속하게 출시하는 것
- 복원력
  - 잘못된 일이 발생하더라도 서비스를 계속 제공
  - 실패가 발생할 때, 처리하고 전체 시스템을 여전히 사용자에게 서비스를 제공
- 확장성
  - 부하에 따라 소프트웨어를 확장할 수 있음
- 비용
  - 필요한 자원을 생성해 실제 사용한 만큼만 지불하고 더 이상 필요 없는 자원은 회수 가능하게함

클라우드 네이티브 구성

클라우드 네이티브 컴퓨팅의 주된 모델은 컨테이너와 서비스리다

클라우드 네이티브 구성

- 컨테이너
  - 가상화 및 컨테이너 기술은 고립된 컨텍스트에서 공유하는 것이 다름.
  - 가상머신은 하드웨어만 공유하고, 컨테이너는 운영체제 커널도 고유함.
  - 컨테이너가 더 가볍고 이식성이 좋음
  - 네임스페이스(Namespace)
    - 프로세스간의 리소스 분할

클라우드 네이티브 구성

- 오케스트레이션
  - CaaS 플랫폼은 여러 서버에서 실행되는 컨테이너를 조정하면서 클라우드 네이티브 환경에서 발생하는 모든 중요한 문제를 해결하기 위해 많은 기능을 제공함
  - 컨테이너의 배포 대상은 서버지만 오케스트레이터의 경우에는 배포 대상이 클러스터임

클라우드 네이티브 구성

- 서버리스 – 개발자가 애플리케이션에 대한 비즈니스 로직을 구현하는 데만 집중. 가상 시스템, 컨테이너, 동적 확장을 포함하여 실행할 애플리케이션에 필요한 모든 인프라 작업을 플랫폼이 대신 처리
  - 서비스형 백엔드(Backend as a service Baas)
    - Okta, Google Firebase
  - 서비스형 함수(Function as a service Faas)
    - Vmware Tanzu application platform, Redhat Openshift Serverless, Google Cloud Run

클라우드 네이티브 애플리케이션을 위한 아키텍처

- 클라우드 네이티브 아키텍처 요소

클라우드 네이티브 애플리케이션을 위한 아키텍처

- 다중 계층에서 마이크로서비스 아키텍처까지 그리고 이후
  - 모놀리스 대 마이크로서비스 모놀리스 아키텍처는 종종 다층 구조로 되어있음
  - 마이크로서비스는 독립적으로 배포할 수 있는 여러 구성 요소로 이루어짐

클라우드 네이티브 애플리케이션을 위한 아키텍처

클라우드 네이티브 애플리케이션을 위한 아키텍처

- 클라우드 네이티브 애플리케이션을 위한 서비스 기반 아키텍처
  - 구축할 시스템의 아키텍처는 서비스와 상호작업에서는 두가지 요소를 가짐
    - 서비스(Service)
    - 상호작용(Interaction)

클라우드 네이티브 애플리케이션을 위한 아키텍처

- 클라우드 네이티브 애플리케이션을 위한 서비스 기반 아키텍처
- 주요 요소는 서로 다른 방식으로 상호작용하는 (애플리케이션 데이터)서비스임

클라우드 네이티브 애플리케이션을 위한 아키텍처

- 애플리케이션 서비스
- 데이터 서비스
- 상호작용

개발 환경 구성 및 설정

https://github.com/Dennis-IDEACUBE/Deploying-Cloud-Native-Applications

- Install WSL & Reboot
- Visual Studio Code & Extensions
  - Install Visual Studio Code
  - Install Visual Studio Code Extensions
    - WSL(or VirtualBox or Vmware)
    - Extension Pack for Java
    - Gradle for Java
    - Spring Boot Extension Pack

개발 환경 구성 및 설정

- Init & Java
  - $ sudo apt install zip
  - $ curl -s "https://get.sdkman.io" | bash
  - $ sdk list java
  - $ sdk install java 17.0.3-tem
  - $ sdk default java 17.0.3-tem
  - $ java --version
  - $ sdk use java 17.0.3-tem

개발 환경 구성 및 설정

  - $ sdk current java
  - $ sudo vi /etc/.bashrc
  - export JAVA_HOME-/home/user1/.sdkman/candidates/java/current
- Docker
  - https://docs.docker.com/engine/install/ubuntu/
    https://docs.docker.com/engine/install/linux-postinstall/
  - $ sudo chmod 666 /var/run/docker.sock
  - $ docker login
  - $ docker pull ubuntu:22.04

개발 환경 구성 및 설정

- Minikube
  - https://minikube.sigs.k8s.io/docs/start/
  - $ minikube start --driver=docker
  - $ minikube config set driver docker
- Kubectl
  - https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/
  - $ kubectl get nodes

개발 환경 구성 및 설정

- $ kubectl get nodes
  - Httpie
  - https://httpie.io/ https://httpie.io/docs/cli/debian-and-ubuntu
- $ http pie.dev/get
  - grype
  - https://github.com/anchore/grype
- Tilt
  - https://docs.tilt.dev/install.html

개발 환경 구성 및 설정

- Octant
  - https://reference.octant.dev/?path=/story/docs-intro--page
- Kubeval
  - https://github.com/instrumenta/kubeval(deprecated)
  - https://github.com/yannh/kubeconform?tab=readme-ov-file
- Knative
  - https://knative.dev/docs/install/quickstart-install/#before-you-begin
- Sources
  - https://github.com/ThomasVitale/cloud-native-spring-in-action/tree/sb-3-main

클라우드 네이티브 패턴 및 기술

- 클라우드 네이티브 개발 원칙
- 스프링을 사용한 클라우드 네이티브 애플리케이션 구축
- 도커를 통한 애플리케이션 컨테이너화
- 쿠버네티스로 컨테이너 관리
- 폴라 북숍: 클라우드 네이티브 애플리케이션

클라우드 네이티브 패턴 및 기술

자바에서 컨테이너를 거쳐 쿠버네티스로 가는 스프링 애플리케이션의 여정

클라우드 네이티브 패턴 및 기술

- 클라우드 네이티브 개발 원칙(Twelve-Factor methodology): 12요소와 확장
- 도커를 통한 애플리케이션 컨테이너화
- 쿠버네티스로 컨테이너 관리
- 폴라 복숍: 클라우드 네이티브 애플리케이션

클라우드 네이티브 개발 원칙: 12요소와 확장

히로쿠 클라우드 플랫폼(Heroku Cloud Platform)의 엔지니어들이 클라우드 네이티브 애플리케이션을 설계하고 구축하기 위한 개발 원칙으로 12요소 방법론은 제안함.

- 클라우드 플랫폼에서 배포하기 적합
- 학장을 염두에 둔 설계
- 다양한 시스템에 적용 가능
- 지속적 배포 및 민첩성을 지원

나중에 이 방법론은 케빈 호프만에 의해 그의 저서 “Beyond the Twelve-Factor App”에서 수정되고 확장되어 원래의 요소들의 내용을 새롭게 추가하고 지금의 15요소 방법론(15-Factor methodology)로 정의됨

클라우드 네이티브 개발 원칙: 12요소와 확장

- 하나의 코드 베이스, 하나의 애플리케이션
- API 우선
- 의존성 관리
- 설계, 빌드, 릴리스, 실행
  - 설계 단계(Design Stage)
  - 빌드 단계(Build Stage)
  - 릴리스 단계(Release Stage)
  - 실행 단계(Run Stage)

클라우드 네이티브 개발 원칙: 12요소와 확장

- 설정, 크리덴셜 및 코드
  - 설정(Configuration)
- 로그
  - 로그 수집기(Log Aggregator)
- 일회성
  - 일회성(Disposability), 우아한 종료(Graceful Shutdown)
- 지원 서비스

클라우드 네이티브 개발 원칙: 12요소와 확장

- 환경 동일성(Environment Parity)
  - 시간 차이(Time Gap)
  - 사람 차이(People Gap)
  - 도구 차이(Tool Gap)
- 관리 프로세스(Administrative Process)
- 포트 바인딩(Port Binding)
- 상태를 갖지 않는 프로세스(Stateless Process)
- 동시성(Concurrency)

클라우드 네이티브 개발 원칙: 12요소와 확장

- 원격 측정(Telemetry)
- 인증 및 승인
  - 제로 트러스트(Zero Trust)
  - OAuth2, OpenID Connect OIDC

스프링을 사용한 클라우드 네이티브 애플리케이션 구축

- 스프링 개요
  - 스프링 프레임워크
    - 스프링 컨텍스트(Spring Context) or 스프링 컨테이너(Spring Container)
  - 스프링 부트

스프링을 사용한 클라우드 네이티브 애플리케이션 구축

- 스프링 부트 애플리케이션 구축
  - C4 모델에 따른 폴라 북숍 애플리케이션의 아키텍처 다이어그램
  - 사람, 시스템 , 컨테이너

도커를 통한 애플리케이션 컨테이너화

- 도커 소개: 이미지 및 컨테이너
  - 도커 서버(Docker Server)
  - 도커 데몬(Docker Daemon)
  - 도커 호스트(Docker host)
  - 도커 클라이언트(Docker Client)
  - 컨테이너 저장소(Container Registry)

도커를 통한 애플리케이션 컨테이너화

- 도커 엔지은 클라이언트/서버 아키텍처를 가지고 있으며 저장소와 상호작용함

- 컨테이너, 이미지

도커를 통한 애플리케이션 컨테이너화

- 컨테이너를 통한 스프링 애플리케이션 실행

쿠버네티스로 컨테이너 관리

컨테이너의 배포 대상은 한 대의 머신이지만, 오케스트레이터는 클러스터임

쿠버네티스로 컨테이너 관리

- 쿠버네티스 소개: 배포, 파드, 서비스
  - 쿠버네티스의 주요 구성 요소는 API, 컨트롤러 프레인, 작업자 노드임

쿠버네티스로 컨테이너 관리

- 쿠버네티스에서 스프링 애플리케이션 실행

쿠버네티스로 컨테이너 관리

폴라 복숍: 클라우드 네이티브 애플리케이션

- 요구사항
  - Polar Bookshop은 북극과 북극 지역에 관한 지식과 정보를 전파하는 전문 서점입니다.
  - 이 서점을 운영하는 기관인 Polarsophia는 온라인을 통해 전 세계에 책을 판매하기로 결정하였고, 이는 그들의 야심찬 프로젝트의 시작에 불과합니다.
  - 조직은 클라우드 네이티브 방식을 도입하여 시스템의 핵심 부분을 구축할 예정이며, 이 시스템은 기능과 통합의 무한한 가능성을 가집니다.
  - 관리 팀은 짧은 반복주기로 새로운 기능을 제공하여 시장 출시 시간을 단축하고 사용자로부터 초기 피드백을 얻을 계획입니다.
  - Polar Bookshop은 고객들이 카탈로그에서 책을 검색하고 구매할 수 있으며, 주문 상태를 확인할 수 있는 애플리케이션을 통해 책을 판매할 예정입니다.
  - 직원들은 책을 관리하고 카탈로그에 새로운 항목을 추가할 수 있습니다.
  - 이 시스템은 여러 서비스로 구성되어 있으며, 비즈니스 로직과 중앙화된 설정 같은 공유 관심사를 구현하는 서비스들을 포함합니다.
  - 이 책에서는 Polar Bookshop의 클라우드 네이티브 시스템 아키텍처와 특정 서비스에 대한 자세한 정보를 제공하고, 시스템을 배포 단계에서 시각화하는 다양한 관점을 채택할 예정입니다.
  - 초기에는 프로젝트에서 사용할 패턴과 기술에 초점을 맞추어 설명할 것입니다.

폴라 복숍: 클라우드 네이티브 애플리케이션

- 프로젝트에서 사용되는 패턴과 기술
  - 웹과 상호작용
  - 데이터
  - 설정
  - 라우팅
  - 관측 가능성
  - 복원력
  - 보안
  - 테스트
  - 빌드 및 배포
  - UI

스프링 부트 컨테이너화

도커에서 컨테이너 이미지로 작업하기

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

도커 컴포즈를 통한 스프링 부트 컨테이너의 관리

배포 파이프라인: 패키지 및 등록

도커에서 컨테이너 이미지로 작업하기

도커 엔진은 클라이언트/서버 아키텍처를 가지고 있고 컨테이너 저장소와 상호 작용함.

도커에서 컨테이너 이미지로 작업하기

- 컨테이너 이미지 이해
  - 컨테이너 이미지는 수정이 불가능한 읽기 전용 레이어를 순서대로 쌓아 올려서 구성됨
  - 첫번째 레이어는 베이스 이미지를 나타내며, 나머지는 그 위에 적용된 수정을 나타냄

도커에서 컨테이너 이미지로 작업하기

- 실행 중에 있는 컨테이너는 이미지 레이어 위에 여분의 레이어가 있음.
- 유일하게 쓸 수 있는 레이어지만 휘발성 있다는 것을 기억해야함

도커에서 컨테이너 이미지로 작업하기

- 도커파일을 통한 이미지 생성
  - 이미지 생성은 도커파일부터 시작됨. 도커파일의 각 명령은 이미지의 레이어를 순서대로 만듬

도커에서 컨테이너 이미지로 작업하기

- 주어진 이름과 버전으로 새 이미지를 빌드하는 도커 CLI 명령

도커에서 컨테이너 이미지로 작업하기

- 깃허브 컨테이너 저장소로 이미지 저장
  - 모든 개인 깃허브 계정에서 사용할 수 있고 공개 저장소에 대해서는 무료임. 비공개 저장소에서도 사용할 수 있지만 제한이 있음
  - 무료 계정으로도 사용료 제한 없이 익명으로 공용 컨테이너 이미지에 접근할 수 있음
  - 깃허브 생태계에 완전히 통합되어 있고 이미지부터 관련 소스 코드까지 원활하게 이동 할 수 있음
  - 무료 계정으로 레지스트리에 엑세스할 수 있는 토큰을 여러 개 생성할 수 있음. 개인 엑세스 토큰(Personal Access Token, PAT)기능 사용할 수 있음

도커에서 컨테이너 이미지로 작업하기

- 컨테이너 이미지는 OCI 호환 컨테이너 저장소의 명명 규칙
  - 컨테이너 저장소(Container Registry)
  - 네이스페이스(Namespace)
  - 이름(Name)과 태크(Tag)

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 스프링 부트의 컨테이너화를 위한 준비
  - 컨테이너 이미지로 패키징 한다는 것은 고립된 콘텍스트에서 계산 리소스와 네트워크를 가지고 애플리케이션을 실행한다는 것을 의미함
    - 어떻게 네트워크를 통해 애플리케이션에 도달할 수 있을까?
    - 어떻게 다른 컨테이너와 연결해 상호 작용할 수 있을까?

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 포트 전달을 사용한 애플리케이션 서비스 노출
  - 포트 매핑을 통해 컨테이너 네트워크에서 외부 세계로의 트래픽을 포워딩 함으로써 컨테이너화된 애플리케이션이 제공하는 서비스에 접근 할 수 있음

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 서비스 발견을 위한 도커 내장 DNS 서버의 사용
  - 카탈로그 서비스 애플리케이션은 포트 매핑 덕분에 PostgreSQL 컨테이너와 상호 작용할 수 있고 외부 세계에서 데이터베이스에 엑세스 할 수 있음

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 카탈로그 서비스 컨테이너는 둘 다 동일한 도커 네트워크에 있기 때문에 PostgreSQL 컨테이너와 직접 상호 작용할 수 있음

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 도커파일로 스프링 부트 컨테이너화
- 프로덕션을 위한 컨테이너 이미지 빌드
    - 성능
        - 계층화된 JAR 모드(Layered-JAR Mode)
            - 의존성(Dependencies)계층
            - 스프링 부트 로더(Spring-boot-loader) 계층
            - 스냅숏 의존성(Snapshot-dependencies)계층
            - 애플리케이션(Application)계층

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 보안
- 도커파일 vs 빌드팩

스프링 부트 애플리케이션을 컨테이너 이미지로 패키지

- 클라우드 네이티브 빌드팩을 이용한 스프링 부트 컨테이너화
  - 클라우드 네이티브 빌드팩에서 제공하는 기능
    - 애플리케이션 유형을 자동으로 감지해 도커파일 없이 패키지를 생성함
    - 다양한 언어와 플랫폼을 지원함
    - 캐시와 레이어를 통해 높은 성능을 갖음
    - 재현 가능한 빌드를 보장함
    - 보안 측면에서 모범 사례를 사용함
    - 프로덕션 환경에 적합한 이미지를 만듬
    - 그랄VM을 사용해 네이티브 이미지의 빌드를 지원함

도커 컴포즈를 통한 스프링 부트 컨테이너의 관리

- 도커 컴포즈를 통한 컨테이너 라이프사이클 관리
- 스프링 부트 컨테이너 디버깅
  - 원하는 대로 컨테이너에서 포트를 노출할 수 있음. 카탈로그 서비스의 경우 서버 포트와 디버그 포트를 모두 노출함

배포 파이프라인: 패키지 및 등록

- 커밋 단계에서 릴리스 후보 빌드
  - 커밋 단계의 마지막에 릴리스 후보를 아티팩트 저장소에 등록함

배포 파이프라인: 패키지 및 등록

- 깃허브 액션을 통한 컨테이너 이미지 등록
  - 깃허브 컨테이너 저장소를 사용하면 컨테이너 이미지가 소스 코드 바로 옆에 출력됨

스프링 부트를 위한 쿠버네티스 기초

도커에서 쿠버네티스로의 이동

스프링 부트를 위한 쿠버네티스 배포

서비스 검색 및 부하 분산

확장성과 일회성

틸트를 사용한 로컬 쿠버네티스 개발

배포 파이프라인: 쿠버네티스 매니페스트 유효성 검사

도커에서 쿠버네티스로의 이동

도커 클라이언트는 도커 호스트라고 부르는 시스템의 리소스만 관리할 수 있는 도커 데몬과 상호 작용함.

애플리케이션은 컨테이너를 통해서 도커 호스트에 배포함.

도커에서 쿠버네티스로의 이동

쿠버네티스를 클라이언트는 쿠버네티스 컨트롤 프레인과 상호작용함. 컨트롤 프레인은 하나 이상의 노드로 구성된 클러스터에서 컨테이너화된 애플리케이션을 관리함. 애플리케이션은 클러스터의 노드에 파드로 배포함

도커에서 쿠버네티스로의 이동

쿠버네티스 구성요소

- 클러스터(Cluster)
- 컨트롤 플레인(Control Plane)
- 작업자 노드(Worker Node)
- 파드(Pod)

도커에서 쿠버네티스로의 이동

- 로컬 쿠버네티스 클러스터
- 로컬 클러스터에서 데이터 서비스 관리

스프링 부트를 위한 쿠버네티스 배포

- 컨테이너에서 파드로
  - 파드는 쿠버네티스에서 배포의 가장 작은 단위임. 적어도 하나의 기본 컨테이너를 실행하고 로깅, 모니터링, 보안과 같은 추가 기능을 위해 선택적으로 핼퍼 컨테이너를 실행할 수 있음

스프링 부트를 위한 쿠버네티스 배포

- 배포를 통한 파드 제어
    - 배포는 클러스터에서 레플리카셋과 파드를 통해 복제된 애플리케이션을 관리함. 레플리카셋은 어떤 상황에서도 원하는 수의 파드가 항상 실행되도록 보장함. 파드는 컨테이너화 된 애플리케이션을 실행함

스프링 부트를 위한 쿠버네티스 배포

- 스프링 부트 애플리케이션을 위한 배포 객체 생성
  - 쿠버네티스 매니페스트는 일반적으로 apiVersion, Kind, metadata, spec의 4가지 주요 섹션으로 구성됨.

서비스 검색 및 부하 분산

- 서비스 검색 및 부하 분산의 이해
  - 베타 앱 인스턴스가 하나만 있으면 알파 앱과 베타 앱 간의 프로세스 간 통신은 DNS이름을 기반으로 하는데 DNS 이름을 통해 베타 앱의 IP 주소를 찾음

서비스 검색 및 부하 분산

- 클라이언트 측 서비스 검색 및 부하 분산
  - 알파 앱과 베타 앱 간의 프로세스 간 통신은 호출할 특정 인스턴스의 IP 주소를 기반으로 하며 서비스 레지스트리에서 조회할 때 반환하는 IP 주소 목록 중에서 하나를 선택함

서비스 검색 및 부하 분산

- 서버 측 서비스 검색 및 부하 분산
  - 알파 앱과 베타 앱 간의 프로세스 간 통신은 DNS 이름을 기반으로 이루어지고 로드 밸런서에 의해 인스턴스 중 하나의 IP 주소로 매핑됨. 서비스 등록 과장은 플랫폼에 의해 처리되고 밖으로 드러나지 않음

서비스 검색 및 부하 분산

- 쿠버네티스에서 알파 앱과 베타 앱 간의 프로세스 간 통신은 서비스 객체를 통해 이루어짐. 서비스에 도착하는 모든 요청은 프록시가 가로챈 다음, 특정 부하 분산 전략에 따라 서비스에 속하는 복제본 중하나로 전달됨

서비스 검색 및 부하 분산

- 쿠버네티스 서비스를 통한 스프링 부트 애플리케이션 노출
  - 클러스터 IP 서비스는 클러스터 내부의 네트워크에 파드 집합을 노출함

서비스 검색 및 부하 분산

- 카탈로그 서비스 애플리케이션은 포트 전달을 통해 로컬 컴퓨터에 노출됨. 카탈로그 서비스와 PostgreSQL은 둘 다 클러스터 내 호스트 이름, IP 주소, 서비스 객체에 할당된 포트를 통해 클러스터 내부에 노출됨.

확장성과 일회성

동일한 애플리케이션의 여러 인스턴스를 배포하는 것이 높은 가용성을 달성하는 데 도움이 됨

워크로드를 다양한 복제본에 분산시킬 수 있으며, 한 인스턴스가 실패하더라도 최소한의 중단으로 교체할 수 있음

15-Factor 방법론에 따라 애플리케이션을 무상태(stateless)이고 일회용(disposable)으로 요구함

확장성과 일회성

일회성을 위한 조건: 빠른 시작

일회성을 위한 조건: 우아한 종료

스프링 부트 애플리케이션 확장

틸트를 사용한 로컬 쿠버네티스 개발

틸트를 사용한 내부 개발 루프

틸트를 사용한 로컬 쿠버네티스 개발

옥탄트를 사용한 쿠버네티스 워크로드 시각화

배포 파이프라인: 쿠버네티스 매니페스트 유효성 검사

커밋 단계에서 쿠버네티스 매니페스트 검증

쿠버네티스 매니페스트가 애플리케이션 저장소에 포함되면 커밋 단계에 새로 포함된 과정에서 유효성을 검사함

감사합니다.