001

1. Tạo Docker Images và Đẩy Lên Docker Hub:

Tạo Docker images cho tất cả 7 ứng dụng với tag "S14".

Đẩy các image này lên Docker Hub để xác nhận, mỗi image đều có tag "s14".

2. Cập Nhật Docker Compose:

Cập nhật file Docker Compose để sử dụng các image "s14" vừa tạo.

Xóa bỏ phần cấu hình cho RabbitMQ và thay thế bằng cấu hình dịch vụ Kafka.

Lấy tham khảo cách cấu hình từ repo GitHub của Bitnami (một nguồn đáng tin cậy được VMware hỗ trợ) để thiết lập Kafka với Docker Compose, bao gồm:

- Định nghĩa service Kafka với image, port mapping và cấu hình volumes (ví dụ: ánh xạ thư mục lưu trữ dữ liệu Kafka vào thư mục trên máy cục bộ).
- Cập nhật các environment variable trong microservice (Accounts, Message) để thay đổi thông tin kết nối từ RabbitMQ sang Kafka (sử dụng biến môi trường spring_cloud_stream_kafka_binder_brokers với giá trị là tên service Kafka và port 9092).

3. Khởi Động Lại Các Container:

Dừng các container và server đang chạy (bao gồm cả Kafka server chạy cục bộ và container Keycloak) để đảm bảo không xung đột.

Chạy lại Docker Compose bằng lệnh docker compose up -d để khởi động tất cả container mới theo cấu hình đã cập nhật.

4. Cấu hình và Kiểm Tra Keycloak:

Cập nhật cấu hình client trong Keycloak (tạo client mới với ID easybank-callcenter-cc, cấu hình xác thực, tắt standard flow và bật service account roles).

Lấy thông tin credential từ Keycloak để dùng trong Postman.

5. Kiểm Tra Giao Tiếp Giữa Microservices:

Dùng Postman gọi API từ Accounts microservice để trigger quá trình giao tiếp bất đồng bộ.

Xác nhận thông qua log của Message microservice (breakpoint dừng tại hàm gửi email, SMS, thông tin offset) và log của Accounts microservice (xác nhận cập nhật trạng thái giao tiếp).

Kiểm tra Kafka topics và consumer thông qua plugin Kafkalytic trong IntelliJ, cho thấy cả hai service (Accounts và Message) kết nối thành công tới Kafka.

6. Tổng Kết và Đánh Giá:

Các thay đổi chỉ bao gồm:

- Thay thế dependency của RabbitMQ bằng Kafka.
- Cập nhật thông tin kết nối trong file application.yml.
- Hệ thống microservices sử dụng Apache Kafka để giao tiếp bất đồng bộ đã được triển khai thành công.

002

Kubernetes là gì?

- Là một hệ thống mã nguồn mở để tự động triển khai, mở rộng và quản lý các ứng dụng container.

Đặc điểm nổi bật:

- Phổ biến nhất hiện nay về orchestration.
- Trung lập với nền tảng đám mây (cloud-neutral): dùng được trên local, AWS, GCP, Azure mà không cần thay đổi nhiều.
- Do Google phát triển, dùng nội bộ hơn 15 năm để vận hành các sản phẩm như YouTube, Gmail, Google Photos... Trở thành mã nguồn mở từ năm 2015.

Lợi ích của Kubernetes:

- Quản lý hệ thống phân tán (microservices, cloud-native).
- Tự động mở rộng, phục hồi khi có lỗi, triển khai không downtime.
- Thay thế Eureka Server: dùng cân bằng tải phía server thay vì client.
- Orchestration cả container và storage.
- Hỗ trợ rollout, rollback, tự phục hồi.
- Quản lý cấu hình và secrets.

Nguồn gốc tên gọi:

- “Kubernetes” bắt nguồn từ tiếng Hy Lạp, nghĩa là “người lái tàu” hoặc “thuyền trưởng” – điều khiển con tàu (các container).
- Logo Kubernetes liên quan đến ý tưởng này.

- Viết tắt thường gặp: K8s (K + 8 chữ cái + s).\

003

1. Tổng quan về Kubernetes và lý do sử dụng

- Kubernetes là một hệ thống orchestration cho containers, không chỉ đơn thuần là công cụ chạy container.
- Docker Compose chỉ triển khai được trong một máy chủ đơn, không phù hợp cho môi trường sản xuất có hàng trăm microservices.
- Kubernetes giúp triển khai microservices trong nhiều node, hỗ trợ tự động hoá: deployment, scaling, rollouts, recovery.

2. Kiến trúc của Kubernetes Cluster

- Kubernetes cluster gồm 2 loại node:
- Master Node (Control Plane): quản lý toàn bộ hệ thống.
- Worker Nodes: nơi các container (microservices) thực sự chạy.

3. Các thành phần chính trong Master Node

- Kube API Server: điểm tiếp nhận lệnh từ người dùng (qua kubectl hoặc UI), xử lý và chuyển tiếp yêu cầu.
- Scheduler: quyết định microservice sẽ được triển khai trên node nào, dựa vào tài nguyên khả dụng.
- Controller Manager: theo dõi trạng thái thực tế vs trạng thái mong muốn. Nếu container chết, nó sẽ tự động tạo lại để khôi phục số lượng replica yêu cầu.
- etcd: cơ sở dữ liệu key-value lưu trữ thông tin cấu hình và trạng thái của cluster. Được coi là "bộ não" của Kubernetes.

4. Các thành phần trong Worker Node

- Kubelet: agent giúp worker node giao tiếp với master node, nhận chỉ thị triển khai.
- Container Runtime: công cụ chạy các container (Docker, containerd,...).

5. Luồng hoạt động khi triển khai một microservice

- Người dùng gửi yêu cầu triển khai (YAML file) qua kubectl.
- Kube API Server nhận lệnh, chuyển tới Scheduler.

- Scheduler chọn worker node phù hợp.
- Kube API Server chuyển lệnh đến Kubelet của node đó.
- Container runtime triển khai microservice.
- Controller Manager giám sát trạng thái để đảm bảo đúng số lượng replica và khôi phục nếu có sự cố.

004

Lý do dùng local cluster trước cloud:

- Tránh phát sinh chi phí cloud khi học.
- Dễ học và thử nghiệm khái niệm Kubernetes.

Tránh dùng Minikube:

- Một số lệnh khác biệt so với môi trường production.
- Giảng viên khuyên dùng Docker Desktop để tạo cụm local Kubernetes giống môi trường thực tế hơn.

Bật Kubernetes trong Docker Desktop:

- Vào Settings → chọn Kubernetes → bật Enable Kubernetes → nhấn Apply & Restart.
- Không bật Show system containers để tránh hiển thị container nội bộ của Kubernetes.

Docker Desktop sẽ tạo cụm single-node:

- Một node vừa làm master vừa làm worker (vì hạn chế tài nguyên).
- Môi trường cloud sau này sẽ dùng multi-node (ít nhất 1 master, 3 worker).

Thiết lập và kiểm tra kubectl:

- kubectl là CLI tương tác với Kubernetes.
- Docker Desktop tự động cài kubectl (nếu không, cần thêm vào PATH).

Kiểm tra:

- kubectl config get-contexts → xem context hiện tại.
- kubectl config get-clusters → liệt kê cluster hiện có.
- kubectl config use-context docker-desktop → chọn đúng context nếu có nhiều context.
- kubectl get nodes → xác nhận cụm có một node đang chạy.

Kết luận:

- Đã thiết lập thành công cụm Kubernetes local.
- Sẵn sàng tiếp tục học và triển khai microservices trong phần tiếp theo.

005

Tạo Service Account:

Một tài khoản admin-user được tạo bằng cách áp dụng file dashboard-adminuser.yml với lệnh:

bash

Sao chép

Chỉnh sửa

kubectl apply -f dashboard-adminuser.yml

Nếu đã tạo trước đó, lệnh sẽ trả về trạng thái unchanged.

Tạo ClusterRoleBinding:

Một file mới tên là dashboard-rolebinding.yaml được tạo để định nghĩa ClusterRoleBinding.

Bên trong, binding này sẽ:

Gắn role cluster-admin (role mặc định trong Kubernetes) cho admin-user.

Xác định rõ kind: ServiceAccount, tên là admin-user, và namespace là kubernetes-dashboard.

Áp dụng file với:

bash

Sao chép

Chỉnh sửa

kubectl apply -f dashboard-rolebinding.yaml

Tạo Token để đăng nhập:

Token được tạo với lệnh:

bash

Sao chép

Chỉnh sửa

kubectl -n kubernetes-dashboard create token admin-user

Token này được dán vào UI của Kubernetes Dashboard để đăng nhập theo lựa chọn "Token".

Sau khi đăng nhập:

Có thể thấy các deployment, pods và replicaset đang chạy trong namespace kubernetes-dashboard.

Thông tin service account admin-user và cluster role binding admin-user cũng hiển thị.

Role cluster-admin cho phép truy cập toàn bộ tài nguyên trong cluster.

Vấn đề của token ngắn hạn:

Token được tạo theo cách trên là ngắn hạn – sẽ hết hạn nếu không sử dụng.

Muốn tạo token dài hạn, cần tạo một Secret với loại kubernetes.io/service-account-token và gán nó cho service account admin-user.

File secret.yaml sẽ được tạo để chứa cấu hình này.

006

Bối cảnh: Kubernetes cluster đã được thiết lập thành công cục bộ (local). Bước tiếp theo là triển khai các microservices, bắt đầu từ Config Server.

Cần làm gì:

- Cung cấp hướng dẫn cho Kubernetes về cách triển khai và expose microservices bằng file cấu hình YAML (Docker Compose không dùng được vì không tương thích với Kubernetes).

Thực hiện:

- Tạo thư mục section_15/Kubernetes trong workspace.
- Mở terminal tại thư mục đó và tạo file configserver.yaml.
- Trong file YAML sẽ có các chỉ dẫn triển khai Config Server.
- Cấu trúc YAML cơ bản:
- apiVersion: apps/v1 và kind: Deployment (bắt buộc, không được viết sai).
- metadata: chứa tên deployment và nhãn (labels).
- spec:
- replicas: số lượng bản sao của pod (ở đây là 1).
- selector: xác định pod cần áp dụng (dựa theo matchLabels).

- template: định nghĩa template pod:
- metadata: nhãn của pod.
- spec: chi tiết container, bao gồm image, port, v.v.

Lưu ý:

- Thực tế, DevOps sẽ chịu trách nhiệm viết file YAML.
- Tuy nhiên, developer cần hiểu cú pháp cơ bản.
- Việc luyện tập thường xuyên sẽ giúp quen thuộc với các file cấu hình Kubernetes, tương tự như Docker Compose.

Deployment Specification:

- Trong phần deployment, một template sẽ được định nghĩa.
- Trong template, thông tin chi tiết như container name, image, và port sẽ được khai báo.

Ví dụ: Image được dùng là eazybytes/configserver:s14, port container là 8071.

Container Registry:

- Nếu không chỉ rõ, Kubernetes mặc định sẽ tìm image trong Docker Hub (docker.io).
- Nếu dùng registry khác (như Amazon ECR), cần ghi rõ đầy đủ đường dẫn.

Định nghĩa container:

- Sử dụng danh sách (-) để định nghĩa nhiều container nếu cần.
- Trong trường hợp này chỉ có một container chính (không cần container phụ trợ).
- Gắn nhãn (Labels):
- Phải thống nhất nhãn (label) như app: configserver giữa deployment và service để Kubernetes ánh xạ đúng.

Phân tách file YAML bằng ---:

- Ba dấu gạch ngang (---) cho phép tách nhiều cấu hình YAML trong một file.
- Dễ dàng hơn trong quản lý khi chỉ cần apply một file thay vì nhiều file.

Service Configuration:

- Kind: Service, API version: v1.

- Tên service là configserver – cần giữ nguyên tên này để các microservice khác có thể giao tiếp.
- Selector sử dụng nhãn app: configserver để liên kết với deployment.
- Type: LoadBalancer dùng để phơi bày service ra bên ngoài cluster.
- Port: 8071, giao thức TCP. Port này sẽ dùng để giao tiếp từ bên ngoài vào container.
- targetPort và containerPort nên giống nhau để tránh lỗi.

Manifest Files:

- Các file YAML này còn được gọi là Kubernetes manifest files – thuật ngữ chuẩn trong ngành.

007

Kiểm tra namespace mặc định

- Sử dụng các lệnh kubectl get deployments, services, replicaset, pods để xác nhận rằng namespace mặc định chưa có tài nguyên nào được tạo.
- Dashboard Kubernetes cũng hiển thị không có deployment, pod hoặc service liên quan đến microservice.

Triển khai Config Server

- Di chuyển tới thư mục chứa file manifest config-server.yaml.
- Chạy lệnh kubectl apply -f config-server.yaml để triển khai.
- Kết quả cho thấy deployment và service được tạo thành công.

Xác nhận triển khai

Dùng các lệnh kubectl get deployments, services, replicaset, pods để kiểm tra trạng thái:

- Deployment có 1 replica đang chạy.
- Service được tạo loại LoadBalancer với IP nội bộ hoặc localhost (nếu chạy local).
- Pod của config server đang ở trạng thái "Running".
- Dashboard Kubernetes hiển thị đầy đủ các thành phần vừa được triển khai.

Kiểm tra hoạt động Config Server

- Truy cập localhost:8071/account/prod, localhost:8071/loans/prod, hoặc localhost:8071/eureka/default để xác nhận các file cấu hình được cung cấp đúng.
- Cấu hình được trả về đúng với từng profile, xác nhận config server hoạt động thành công.

Kết luận

- Việc triển khai config server lên Kubernetes đã thành công và có thể truy cập được thông qua địa chỉ localhost và port đã chỉ định (8071).

008

Lý do cần ConfigMap:

- Trong Docker Compose, các container phụ thuộc vào nhiều biến môi trường (ví dụ: profile, config server URL, Eureka server URL, Keycloak…).
- Khi triển khai lên Kubernetes, cần tạo các biến môi trường tương tự để microservices hoạt động đúng cách.

Sử dụng ConfigMap:

- Tạo file configmaps.yaml.
- Khai báo thông tin cơ bản như apiVersion: v1, kind: ConfigMap, metadata.name, và phần data chứa các cặp key-value là các biến môi trường.

Ví dụ các biến môi trường được khai báo:

- SPRING_PROFILES_ACTIVE
- SPRING_CONFIG_IMPORT
- EUREKA_CLIENT_SERVICEURL_DEFAULTZONE
- Các tên ứng dụng như config server, Eureka, accounts, loans, gateway…
- Thông tin về Keycloak: username, password, URL.

Lưu ý:

- Các host/service name trong biến môi trường phải trùng với tên service khai báo trong Kubernetes.

- Không khai báo biến môi trường cho Kafka, RabbitMQ hay OpenTelemetry ở bước này vì sẽ sử dụng Helm Chart cho các công cụ này.

Tạo ConfigMap:

- Chạy lệnh: kubectl apply -f configmaps.yaml
- Kiểm tra trong Kubernetes Dashboard → mục "Config Maps".
- So sánh ConfigMap và Secrets:
- Dữ liệu trong ConfigMap hiển thị rõ ràng.
- Secrets được mã hóa base64 và ẩn trong dashboard.
- Secrets không thực sự an toàn tuyệt đối – nên dùng giải pháp bảo mật cao hơn trong môi trường cloud.

Kết luận:

- Sau khi tạo ConfigMap, có thể triển khai các microservices lên Kubernetes.
- Keycloak là bắt buộc vì các API được bảo vệ bởi OAuth2.

009

Mục tiêu: Triển khai toàn bộ các microservices còn lại vào Kubernetes cluster.

Chuẩn bị:

- Tác giả đã chuẩn bị sẵn các file manifest YAML trong thư mục Section 15.
- Đặt tên file theo số thứ tự (1, 2, 3...) để đảm bảo thứ tự triển khai chính xác.

Thứ tự triển khai đề xuất:

- Keycloak
- ConfigMap
- Config Server
- Eureka Server
- Accounts
- Loans
- Cards
- Gateway

Chi tiết file keycloak.yaml:

- Gồm deployment và service.
- Chạy ở chế độ dev với tham số start-dev.

- Sử dụng biến môi trường từ ConfigMap (qua env -> valueFrom -> configMapKeyRef).
- Expose service kiểu LoadBalancer ở port 7080 (nội bộ container dùng 8080).

Chi tiết file EurekaServer.yaml:

- Dùng image có tag S12 để không cần Kafka hoặc RabbitMQ.
- Có 2 biến môi trường: tên app và URL của config server.

Chi tiết các microservices khác (accounts.yaml, loans.yaml, cards.yaml, gateway.yaml):

- Tất cả đều dùng image với tag S12.
- Đều khai báo biến môi trường như Eureka URL để đăng ký dịch vụ.
- Gateway có thêm biến môi trường để trỏ tới Keycloak server URL.

Lưu ý quan trọng:

- Biến môi trường trong env cần khớp với tên dùng trong ứng dụng.
- Có thể dùng tên key khác trong ConfigMap, nhưng nên giữ nhất quán để dễ quản lý.

Kết luận:

- Tất cả manifest files đã được chuẩn bị và giải thích rõ.
- Bài học tiếp theo sẽ áp dụng các YAML này để triển khai vào Kubernetes cluster.

010

Triển khai Keycloak:

Sử dụng lệnh kubectl apply -f 1_keycloak.yaml để tạo dịch vụ Keycloak.

ConfigMap:

Dùng configmaps.yaml để tạo lại ConfigMap nhằm minh họa Kubernetes phát hiện không có thay đổi và trả về trạng thái unchanged.

Triển khai Config Server:

Thay đổi image từ S14 sang S12, triển khai lại để minh họa cách Kubernetes cập nhật deployment nhưng giữ nguyên service nếu không đổi.

Xác minh Config Server:

Kiểm tra logs trên Kubernetes Dashboard để xác nhận đã triển khai thành công.
Triển khai Eureka Server:
Áp dụng file manifest, theo dõi quá trình start và kiểm tra logs để xác nhận hoàn tất.
Triển khai các Microservices:
Gồm accounts, loans, cards lần lượt từ các file 5_accounts.yaml, 6_loans.yaml, 7_cards.yaml.
Phải chờ các container này khởi động xong trước khi triển khai Gateway.
Lưu ý về thứ tự khởi động:
Kubernetes không hỗ trợ depends_on như Docker Compose, do đó cần đảm bảo các dịch vụ phụ thuộc khởi động trước.
Kiểm tra trên Eureka:
Đảm bảo accounts, loans, cards đã đăng ký thành công vào Eureka Server.
Triển khai Gateway Server:
Triển khai Gateway sau khi các dịch vụ con sẵn sàng.
Cấu hình Keycloak để lấy token:
Truy cập Keycloak qua cổng 7080, tạo client easybank-callcenter-cc, thiết lập các quyền (accounts, cards, loans) và gán vào client.
Test với Postman:
Dùng client secret để lấy access token, gửi các request tạo tài khoản/cards/loans.
Cuối cùng, test API fetchCustomerDetails để xác minh toàn bộ hoạt động.
Kết luận:
Tất cả microservices hoạt động tốt trên Kubernetes.
Mặc dù chưa thấy điểm khác biệt rõ so với Docker Compose, phần "ma thuật" của Kubernetes sẽ được giới thiệu ở các bài sau.

011
Triển khai các microservices vào cụm Kubernetes đã hoàn tất.

Kubernetes có khả năng tự phục hồi – khi một container gặp lỗi hoặc bị tắt, nó sẽ tự động tạo lại container đó để đảm bảo trạng thái luôn khớp với trạng thái mong muốn (desired state).

Thực hiện demo:

Kiểm tra số lượng bản sao bằng kubectl get replicaset.

Thay đổi file YAML để tăng số lượng bản sao (replicas) cho accounts microservice.

Áp dụng thay đổi bằng kubectl apply -f 5_accounts.yaml.

Kiểm tra lại bằng kubectl get pods để xác nhận có thêm pod mới được tạo.

Xóa thủ công một pod, sau đó quan sát Kubernetes tự tạo lại pod mới để duy trì số lượng replicas.

Dùng kubectl get events để kiểm tra log sự kiện, thấy rõ pod bị killed và pod mới được tạo ngay sau đó.

Kết luận:

Kubernetes liên tục giám sát trạng thái của container.

Khi container bị lỗi, nó sẽ thực hiện hành động để khôi phục lại số lượng mong muốn.

Docker hoặc Docker Compose không có khả năng này, vì vậy Kubernetes là lựa chọn ưu tiên để quản lý container trong môi trường production.

012

1. Giới thiệu về tính năng

Bài giảng hướng dẫn cách triển khai các thay đổi mới vào cụm Kubernetes mà không gây gián đoạn dịch vụ, và cách rollback nếu có sự cố xảy ra.

2. Giảm số lượng replica để tiết kiệm tài nguyên

Sử dụng lệnh kubectl scale deployment để điều chỉnh số lượng replica của microservice (accounts-deployment) từ 2 về 1.

Có hai cách: thay đổi trực tiếp trong YAML hoặc dùng lệnh CLI.

Nên cập nhật file YAML để tránh lỗi khi deploy lại sau này.

3. Triển khai phiên bản mới của Gateway Server

Kiểm tra image hiện tại là gatewayserver:s12.

Mục tiêu: deploy lại phiên bản không có security (s11) để loại bỏ yêu cầu token khi gọi API.

Cách làm:

Dùng lệnh kubectl set image để cập nhật image trong deployment.

Đầu tiên thử với image sai (s111) để minh họa xử lý lỗi: Kubernetes không xoá pod cũ cho tới khi pod mới hoạt động ổn định.

Sau đó cập nhật đúng image s11, pod mới được tạo thành công và pod cũ bị xóa.

4. Cơ chế update không downtime

Kubernetes triển khai các pod mới từng cái một.

Khi pod mới chạy ổn, Kubernetes mới xoá pod cũ, đảm bảo không bị downtime.

Phù hợp với hệ thống có nhiều instance.

5. Kiểm tra hoạt động của pod mới

Dùng Postman để gọi API không cần token => thành công, chứng tỏ image không có security đã được deploy.

6. Rollback nếu có lỗi với bản mới

Dùng kubectl rollout history để xem lịch sử triển khai.

Có thể rollback lại bản ổn định trước đó (ví dụ từ s11 quay lại s12).

013

Hiện tại, tất cả các microservices đều được triển khai với type là LoadBalancer, điều này làm cho tất cả các service đều có thể truy cập từ bên ngoài cluster.

Đây là cách tiếp cận không đúng, vì tất cả các yêu cầu từ bên ngoài nên đi duy nhất qua API Gateway — đóng vai trò như một edge server, còn các microservices khác chỉ nên truy cập nội bộ.

Kubernetes cung cấp 3 loại service phổ biến:

ClusterIP (mặc định): chỉ cho phép truy cập nội bộ trong cluster. Dùng khi không muốn expose ra ngoài.

NodePort: expose service tại một port ngẫu nhiên (30000-32767) của mỗi worker node. Có thể truy cập từ bên ngoài, nhưng phụ thuộc vào IP node – dễ gặp vấn đề khi node thay đổi.

LoadBalancer: bên ngoài sẽ được cấp một public IP cố định, client chỉ cần gửi traffic đến đó. LoadBalancer sẽ định tuyến traffic vào trong cluster thông qua NodePort → ClusterIP.

Giải pháp đúng:

Chỉ Gateway Service nên dùng LoadBalancer để nhận traffic từ bên ngoài.

Các service khác (Accounts, Loans, Cards...) nên dùng ClusterIP, giúp bảo mật và giảm rò rỉ thông tin.

Khi dùng ClusterIP:

Pod có thể scale lên nhiều replica.

K8s sẽ tự động load balance yêu cầu nội bộ.

Service được truy cập qua ClusterIP hoặc service name tại cổng được chỉ định (ví dụ: Port 80 → targetPort 8080).

NodePort thì cho phép truy cập từ bên ngoài nhưng phải biết IP của node – điều này không linh hoạt khi node bị thay thế.

LoadBalancer là lựa chọn tốt hơn NodePort nếu cần truy cập từ ngoài, nhưng vẫn không nên dùng nó cho tất cả các service — chỉ dùng cho Gateway.

014

Demo về các loại Kubernetes Service

Kiểm tra dịch vụ:

Sử dụng lệnh kubectl get services để xem các service hiện có.

Ví dụ: accounts microservice có type: LoadBalancer, và vì chạy trên local nên external IP là localhost.

Giải thích từng loại Service:

LoadBalancer:

Tạo IP bên ngoài (trong môi trường cloud là IP public).

Truy cập được từ bên ngoài (ví dụ: localhost:8080/api/contact-info).

Kubernetes sẽ load balance yêu cầu đến các pod.

Lưu ý: Cloud sẽ tính phí cho IP public, và có nguy cơ bảo mật nếu mở trực tiếp.

ClusterIP:

Chỉ cho phép truy cập từ bên trong cluster.

Không thể truy cập bằng trình duyệt từ bên ngoài.
Thích hợp khi dùng kết hợp với API Gateway hoặc Edge Server.
NodePort:
Tạo một cổng mở trên mỗi node (ví dụ: 31182).
Có thể truy cập từ bên ngoài qua IP của node và port này.
Không ổn định nếu microservice di chuyển sang node khác.
Khuyến nghị:
Không nên dùng LoadBalancer cho từng microservice vì:
Tăng chi phí (mỗi service có một LoadBalancer).
Giảm bảo mật.
Nên để type: ClusterIP cho hầu hết các service.
Chỉ để type: LoadBalancer cho Gateway Server để tiếp nhận yêu cầu từ bên ngoài.
Kết luận:
Đã demo cách chuyển đổi giữa LoadBalancer, ClusterIP và NodePort.
Trong thực tế, nên dùng ClusterIP kết hợp với một API Gateway để tiếp nhận lưu lượng từ bên ngoài.

015

Trong phần này, tác giả trình bày việc triển khai các microservices vào Kubernetes bằng cách sử dụng các Kubernetes manifest files. Tuy nhiên, cách tiếp cận này có nhiều hạn chế và vấn đề, đặc biệt khi triển khai ở quy mô lớn:
Khó quản lý khi số lượng microservices lớn: Việc tạo và duy trì hàng trăm manifest file cho từng microservice sẽ rất phức tạp và mất thời gian.
Khó áp dụng cho nhiều môi trường: Với các môi trường khác nhau như Dev, QA, Prod — mỗi môi trường có yêu cầu cấu hình khác nhau (số replica chẳng hạn) — dẫn đến việc phải tạo nhiều bản manifest tương ứng.
Triển khai thủ công: Cần phải chạy lệnh kubectl apply -f cho từng file, rất tốn công.
Gỡ bỏ cũng phức tạp: Khi cần xóa các microservices, phải chạy lệnh kubectl delete -f thủ công cho từng file.

Giải pháp được giới thiệu là Helm – một trình quản lý package cho Kubernetes, giúp tự động hóa việc cài đặt, cập nhật, cấu hình và gỡ bỏ ứng dụng dễ dàng hơn. Tác giả sẽ tiếp tục hướng dẫn về Helm trong phần tiếp theo.
Lưu ý: Mã nguồn trong phần này không thay đổi so với trước, chỉ có các manifest files trong thư mục section15.