ỨNG DỤNG CLOUD NATIVE

1. Giới thiệu Trong phần này, chúng ta sẽ tìm hiểu về ứng dụng Cloud Native, bao gồm định nghĩa, các đặc điểm chính, phương pháp 12 Factor và 15 Factor. Đây là những tiêu chuẩn quan trọng đối với các nhà phát triển Microservices, góp phần xây dựng các hệ thống mành lề, dễ quản lý và mở rộng linh hoạt.

2. Định nghĩa Cloud Native Applications Cloud Native Applications là các ứng dụng được thiết kế và phát triển để tối ưu hóa cho các môi trường điện toán đám mây (Cloud Computing). Nhờ tính linh hoạt, khả năng mở rộng và tính đàn hồi, các ứng dụng này tỪdn dụng tối đa các dịch vụ và công nghệ tài trợ do các nhà cung cấp điện toán đám mây như AWS, Azure, GCP cung cấp.

3. Đặc điểm của Cloud Native Applications

- **Scalability (Khả năng mở rộng)**: Dễ dàng tăng hoặc giảm tài nguyên khi nhu cầu thay đổi.
- **Elasticity (Tính linh hoạt)**: Hệ thống tự điều chỉnh theo tải.
- **Resilience (Tính bền vững)**: Duy trì hoạt động ngay cả khi xảy ra sự cố.
- **Manageability (Dễ quản lý)**: Tích hợp tự động hoá và giám sát.
- **Observability (Tính quan sát)**: Hệ thống cung cấp thông tin chi tiết về hoạt động.

4. Phương pháp 12 Factor và 15 Factor

- **12 Factor Methodology**: Được dùng để xây dựng các ứng dụng dễ bảo trì, linh hoạt, bao gồm các yếu tố như codebase duy nhất, tự động hoá cài đặt, quản lý cấu hình theo môi trường, v.v.
- **15 Factor Methodology**: Mở rộng từ 12 Factor, bao gồm thêm các yếu tố như tính quan sát, quản lý trình, tính an toàn và đồng bộ API.

5. Liên hệ giữa Cloud Native Applications và Microservices Cloud Native Applications và Microservices thường được kết hợp để xây dựng hệ thống linh hoạt, tự động và khả dễ nâng cấp. Các Microservices được thiết kế theo kiểu Cloud Native giúp giảm rủi ro, nâng cao tính bền vững và hiệu suất.

Đặc điểm Quan Trọng Của Ứng Dụng Cloud Native

Hiện tại, chúng ta đã hiểu rõ khái niệm về các ứng dụng Cloud Native. Trong báo cáo này, chúng ta sẽ tìm hiểu các đặc điểm quan trọng của chúng.

1. Microservices

Đặc điểm đầu tiên và quan trọng nhất của các ứng dụng Cloud Native là microservices. Việc phát triển các ứng dụng dựa trên kiến trúc microservices giúp chia nhỏ các thành phần công việc, tạo sự linh hoạt trong quá trình phát triển, triển khai và mã hóa độc lập.

2. Containerization

Sau khi xây dựng microservices, các ứng dụng thường được container hóa bằng Docker hoặc các phần mềm khác. Các container giúp tạo môi trường nhẹ nhàng, nhất quán, và dễ triển khai trên nhiều nền tảng đám mây khác nhau như AWS, GCP, và Azure.

3. Khả năng Mã hóa và Triển khai Linh hoạt

Các ứng dụng Cloud Native hỗ trợ khả năng mã hóa và triển khai linh hoạt, cho phép dễ dàng mở rộng theo chiều ngang để đáp ứng lưu lượng truy cập. Việc thêm instance dịch vụ diễn ra một cách tự động nhờ vào các nền tảng điều phối container như Kubernetes.

4. Tuân thủ Nguyên tắc DevOps

Các ứng dụng Cloud Native áp dụng các nguyên tắc DevOps để tối ưu hóa quá trình phát triển và triển khai. Bằng việc kết hợp CI/CD (Continuous Integration/Continuous Delivery), đội ngũ phát triển và vận hành có thể làm việc hiệu quả hơn, giảm thiểu xung đột và nâng cao tốc độ triển khai.

5. Khả năng Chịu lực và Hồi phục Sau Sự cố

Các ứng dụng Cloud Native được thiết kế với tính chịu lực và hồi phục nhanh chóng khi xảy ra sự cố. Chúng sử dụng các kỹ thuật như cân bằng tải, hồi phục tự động, và triển khai ở nhiều vùng khác nhau để đảm bảo tính sẵn sàng cao.

6. Tối ưu Hóa Dịch vụ Đám mây

Các ứng dụng Cloud Native khai thác tối đa các dịch vụ đám mây như các dịch vụ lưu trữ, điều phối và bảo mật. Như vậy, doanh nghiệp và nhà phát triển có thể tập trung vào logic ứng dụng mà không cần quá quan tâm đến cấu hình hạ tầng.

7. So Sánh Ứng Dụng Cloud Native và Ứng Dụng Truyền Thống

7.1. Hành vi dự báo

- **Cloud Native**: Ứng dụng Cloud Native có hành vi dự báo cao, nhờ việc chia nhỏ logic kinh doanh thành các microservices. Khi xảy ra sự cố, người phát triển có thể nhanh chóng xác định và khắc phục.
- **Truyền Thống**: Ứng dụng truyền thống có logic kinh doanh tập trung, gây khó khăn trong việc xác định nguồn gốc lỗi, dẫn đến thời gian khắc phục lâu hơn.

7.2. Phụ thuộc hệ điều hành

- **Cloud Native**: Hoạt động độc lập với hệ điều hành nhờ Docker và các công nghệ container khác.
- **Truyền Thống**: Phụ thuộc vào hệ điều hành, dẫn đến việc di chuyển hoặc triển khai trên nhiều nền tảng khác nhau trở nên khó khăn.

7.3. Quy mô và khả năng tăng trưởng

- **Cloud Native**: Quy mô linh hoạt và có thể mở rộng dễ dàng do các microservices hoạt động độc lập.
- **Truyền Thống**: Quy mô lớn, các thành phần phụ thuộc lẫu dẫn đến khó khăn trong việc tăng trưởng.

7.4. Quy trình phát triển

- **Cloud Native**: Hỗ trợ Continuous Delivery và DevOps, giúp tối ưu quy trình triển khai và phát triển.

- **Truyền Thống**: Theo mô hình thác nước, dẫn đến thời gian phát triển lâu hơn và không linh hoạt.

7.5. Khả năng phục hồi và tăng trưởng tự động

- **Cloud Native**: Hỗ trợ khả năng phục hồi nhanh chóng và tăng trưởng tự động nhờ Kubernetes.
- **Truyền Thống**: Phục hồi chậm, yêu cầu quá trình giám sát và can thiệp thủ công.

Phương Pháp 12 Factor và 15 Factor

2.1. 12 Factor Methodology

Vào năm 2012, đội ngũ kỹ sư của nền tảng Heroku đã giới thiệu phương pháp 12 Factor nhằm giúp các nhà phát triển xây dựng ứng dụng Cloud Native một cách hiệu quả. 12 nguyên tắc này bao gồm:

1. **Codebase** – Mỗi ứng dụng chỉ có một repository duy nhất.
2. **Dependencies** – Quản lý phụ thuộc một cách rõ ràng.
3. **Config** – Lưu cấu hình bên ngoài code.
4. **Backing Services** – Coi các dịch vụ bên ngoài như cơ sở dữ liệu, message queue là tài nguyên đính kèm.
5. **Build, Release, Run** – Tách biệt rõ ràng quá trình build, release và run.
6. **Processes** – Ứng dụng chạy dưới dạng các tiến trình không trạng thái.
7. **Port Binding** – Ứng dụng tự đóng gói và hoạt động thông qua cổng riêng.
8. **Concurrency** – Tận dụng mô hình process để mở rộng ứng dụng.
9. **Disposability** – Ứng dụng có thể khởi động nhanh và tắt gọn gàng.
10. **Dev/Prod Parity** – Môi trường phát triển và sản xuất cần nhất quán.
11. **Logs** – Ghi log dưới dạng sự kiện có thể phân tích.
12. **Admin Processes** – Chạy các tiến trình quản trị như một phần của ứng dụng.

Việc tuân theo phương pháp 12 Factor giúp các ứng dụng dễ triển khai, mở rộng và quản lý hiệu quả trên các nền tảng đám mây.

2.2. Sự Mở Rộng Thành 15 Factor

Mặc dù 12 Factor đã mang lại nhiều lợi ích, nhưng sự tiến bộ của công nghệ đòi hỏi các nguyên tắc này cần được cập nhật. Do đó, Kevin Hoffman đã mở rộng phương pháp này lên thành 15 Factor bằng cách bổ sung ba nguyên tắc mới:

13. **Telemetry** – Thu thập dữ liệu giám sát để theo dõi sức khỏe hệ thống.
14. **Authentication & Authorization** – Tăng cường bảo mật thông qua quản lý xác thực và phân quyền.
15. **API First** – Xây dựng ứng dụng theo mô hình API trước để tăng khả năng tích hợp.

Phương pháp 15 Factor không chỉ kế thừa các nguyên tắc cốt lõi từ 12 Factor mà còn cải thiện khả năng theo dõi, bảo mật và tích hợp hệ thống.

3. Lợi Ích Khi Áp Dụng 12 Factor và 15 Factor

Việc tuân theo các phương pháp này giúp các ứng dụng Cloud Native đạt được:

- **Khả năng triển khai linh hoạt** trên nhiều nền tảng đám mây.
- **Tính mở rộng** dễ dàng khi nhu cầu hệ thống tăng.
- **Tính di động** giữa các môi trường mà không cần điều chỉnh lớn.
- **Bảo mật và giám sát tốt hơn**, giúp giảm rủi ro trong vận hành.
- **Hỗ trợ Continuous Deployment**, giúp cải thiện quy trình phát triển và triển khai phần mềm.

Các nguyên tắc trong phương pháp 15 Factor

1. Nguyên tắc 1: Một codebase cho một ứng dụng

Nguyên tắc này yêu cầu mỗi ứng dụng hoặc microservice phải có một codebase duy nhất trong hệ thống quản lý phiên bản (VCS). Việc duy trì codebase riêng biệt giúp quản lý và tổ chức code rõ ràng hơn. Trong trường hợp các microservices có chung mã nguồn, những thành phần chung nên được quản lý tạch biệt dưới dạng thư viện hoặc service hậu thuẬd.

2. Nguyên tắc 2: API First

Nguyên tắc API First khuyên khích xây dựng ứng dụng với tư duy API ngay từ giai đoạn thiết kế. Việc thiết kế hệ thống theo API giúp tăng tính linh hoạt, dễ dàng tích hợp và mở rộng. Ngoài ra, việc sử dụng API giúp nhiều đội nhóm có thể phát triển song song các tính năng khác nhau mà không gây xung đột.

3. Nguyên tắc 3: Quản lý phụ thuộc

Mỗi ứng dụng phải khai báo rõ ràng các phụ thuộc trong một tệp tin duy nhất (manifest file) và quản lý chúng bằng các công cụ quản lý phụ thuộc như Maven hoặc Gradle (trong Java). Việc tự động hoá việc quản lý phụ thuộc giúp đảm bảo ổn định và dễ dàng triển khai hệ thống.

4. Nguyên tắc 4: Thiết kế, Xây dựng, Phát hành, Chạy

Quy trình phát triển ứng dụng nên tuân thủ các giai đoạn:

- **Thiết kế**: Xác định công nghệ, công cụ, phụ thuộc cần thiết.
- **Xây dựng**: Biên dịch và đóng gói ứng dụng.
- **Phát hành**: Gán phiên bản và chuẩn bị cho triển khai.
- **Chạy**: Vận hành ứng dụng trên nhiều môi trường khác nhau.

Việc tuân thủ nguyên tắc này giúp tăng tính ổn định và khả năng bảo trì.

5. Nguyên tắc 5: Quản lý cấu hình tách biệt

Cấu hình ứng dụng không nên được lưu trữ trong mã nguồn mà phải được tách biệt và quản lý bằng biến môi trường hoặc kho lưu trữ cấu hình. Điều này giúp dễ dàng thay đổi cấu hình theo môi trường mà không cần chỉnh sửa code.

6. Nguyên tắc 6: Xử lý trạng thái theo nguyên tắc chia sẻ ít nhất

Ứng dụng cần thiết kế sao cho stateless (không trạng thái) khi có thể, nghĩa là không lưu trạng thái cục bộ trên máy chủ mà phải sử dụng cơ chế lưu trữ bên ngoài như cơ sở dữ liệu hoặc bộ nhớ đệm.

7. Nguyên tắc 7: Quản lý tiến trình như thể hiện riêng lẻ

Ứng dụng nên được thiết kế để chạy dưới dạng nhiều tiến trình riêng biệt thay vì một khối lớn duy nhất. Điều này giúp dễ dàng mở rộng theo chiều ngang và quản lý tài nguyên hiệu quả hơn.

8. Nguyên tắc 8: Ràng buộc cổng dịch vụ

Ứng dụng không nên phụ thuộc vào một máy chủ cụ thể mà nên tự đảm nhận việc ràng buộc cổng dịch vụ. Điều này giúp dễ dàng di chuyển và triển khai ứng dụng trên nhiều nền tảng khác nhau.

9. Nguyên tắc 9: Đồng nhất giữa môi trường phát triển và môi trường sản xuất

Ứng dụng nên được phát triển và kiểm thử trong môi trường càng giống với môi trường sản xuất càng tốt. Điều này giúp giảm thiểu lỗi phát sinh khi triển khai thực tế.

10. Nguyên tắc 10: Khai báo và tự động hóa quy trình triển khai

Việc triển khai ứng dụng nên được thực hiện thông qua các quy trình khai báo rõ ràng và tự động hóa bằng các công cụ CI/CD để đảm bảo tính nhất quán và giảm thiểu rủi ro.