High Availability

Redis	Redis Cluster Redis Cluster là một mô hình khác của Redis, tập trung vào phân tán dữ liệu và mở rộng quy mô. Thay vì chỉ có một Master với các Slave, Cluster chia dữ liệu thành các slot, mỗi node chịu trách nhiệm quản lý một phần hoặc nhiều phần của dữ liệu. Khi một node gặp sự cố, Cluster có thể thực hiện failover tự động dựa trên cơ chế replica giữa các node. Khác với Sentinel, mục tiêu chính của Cluster không chỉ là HA mà còn là scale-out để xử lý lượng dữ liệu lớn và tải cao. Cấu hình Cluster phức tạp hơn do phải xác định các slot, replica, và đảm bảo tất cả node đồng bộ dữ liệu đúng cách. Redis Cluster phù hợp cho các ứng dụng cần phân tán dữ liệu, có khả năng mở rộng linh hoạt, và vẫn đảm bảo tính HA thông qua replica của từng node.
	Redis Sentinel Redis Sentinel là một giải pháp của Redis để đảm bảo High Availability (HA) cho hệ thống. Mục tiêu chính của Sentinel là giám sát các Redis Master/Slave, phát hiện khi Master gặp sự cố và tự động đề cử một Slave trở thành Master mới. Sentinel theo dõi trạng thái các node, xác nhận lỗi thông qua quorum (số lượng node cần đồng thuận) trước khi tiến hành failover, từ đó giảm thiểu rủi ro mất dữ liệu và đảm bảo hệ thống luôn sẵn sàng. Cấu hình Redis Sentinel bao gồm việc xác định Master, port của Sentinel, mật khẩu để Sentinel xác thực với Master, thời gian failover tối đa và số lượng Slave đồng bộ song song khi Master bị thay thế. Khi một Master gặp lỗi, Sentinel sẽ yêu cầu các Slave còn lại xác nhận tình trạng và sau đó thực hiện failover. Sau khi failover hoàn tất, các Slave sẽ đồng bộ dữ liệu từ Master mới, còn node cũ khi quay lại sẽ trở thành Slave của Master hiện tại. Sentinel thích hợp với các hệ thống cần HA đơn giản, triển khai nhanh chóng và quản lý tập trung. Các lưu ý quan trọng khi triển khai Sentinel là đảm bảo Redis Master-Slave đã được cài đặt và hoạt động đồng bộ, mật khẩu và các thông số cấu hình phải nhất quán trên tất cả node. Ngoài ra, các thiết lập nâng cao của Redis như appendonly hay appendfsync cũng ảnh hưởng đến độ bền dữ liệu, nên tùy theo yêu cầu của dự án mà bật hoặc điều chỉnh để cân bằng giữa hiệu năng và an toàn dữ liệu.
	So sánh Redis Sentinel và Redis Cluster Redis Sentinel và Redis Cluster phục vụ những mục đích khác nhau, mặc dù cả hai đều hỗ trợ failover. Sentinel tập trung vào HA với kiến trúc Master-Slave đơn giản, dễ triển khai, còn Cluster hướng tới phân tán dữ liệu và mở rộng quy mô, cấu hình phức tạp hơn. Trong thực tế, Sentinel thường được ưa chuộng cho các hệ thống cần triển khai nhanh, dữ liệu vừa phải, còn Cluster thích hợp cho các hệ thống lớn, dữ liệu phân tán và yêu cầu khả năng mở rộng linh hoạt.
PostgreSQL	Giới thiệu về HA PostgreSQL High Availability (HA) là một yêu cầu quan trọng đối với các hệ thống cơ sở dữ liệu doanh nghiệp, đảm bảo dữ liệu luôn sẵn sàng và hệ thống chịu được sự cố của một hoặc nhiều node. PostgreSQL có thể được triển khai HA bằng nhiều mô hình khác nhau, trong đó phổ biến nhất là sử dụng PGPool hoặc kết hợp Patroni, Etcd và HAProxy. Mô hình Patroni được ưa chuộng hơn vì cung cấp cơ chế failover tự động, dễ dàng quản lý cluster, và hỗ trợ mở rộng node linh hoạt mà không gây gián đoạn dịch vụ.
	Kiến trúc tổng thể Một cụm HA PostgreSQL với Patroni thường bao gồm ba thành phần chính: - Patroni: quản lý cluster PostgreSQL, thực hiện failover khi node Primary gặp sự cố và đảm bảo các replica đồng bộ. - Etcd: lưu trữ trạng thái cluster dưới dạng key-value, cung cấp thông tin về leader và các node hiện tại, giúp Patroni phối hợp các node trong cluster. - HAProxy: hoạt động như load balancer, cung cấp điểm truy cập duy nhất cho client, phân tách các kết nối read/write và read-only, giúp client luôn kết nối đúng node mà không cần biết role thực sự của từng node. Cluster thường triển khai tối thiểu ba node để đảm bảo quorum. Client kết nối tới HAProxy, từ đó HAProxy điều hướng các truy vấn read/write đến Primary hoặc replica một cách tự động.
	Triển khai PostgreSQL và cài đặt cơ bản Trước khi triển khai HA, các server cần được chuẩn bị sẵn: cài đặt PostgreSQL (thường dùng Percona PostgreSQL) và kiểm tra trạng thái dịch vụ. Các gói bổ trợ Python cũng cần được cài để Patroni và Etcd có thể tương tác với PostgreSQL. Trước khi Patroni khởi tạo cluster, dữ liệu cũ của PostgreSQL nên được xóa để tránh xung đột với cluster mới.
	Cấu hình Etcd Etcd đóng vai trò lưu trữ trạng thái cluster. Node đầu tiên được khởi tạo với trạng thái new, chỉ định tên node, token cluster, và thư mục lưu dữ liệu riêng biệt. Các node tiếp theo sẽ join cluster bằng lệnh etcdctl member add, cập nhật trạng thái existing và khởi động dịch vụ. Việc triển khai Etcd theo phương pháp này giúp dễ dàng mở rộng cluster bằng cách thêm các node mới sau này mà không làm gián đoạn cluster hiện tại.

	Cấu hình Patroni Patroni được cấu hình qua file patroni.yml, trong đó định nghĩa các thông số: tên node, IP, thư mục dữ liệu, namespace, scope, và tham chiếu đến cluster Etcd. Ngoài ra, các tham số PostgreSQL nâng cao như max_wal_size, work_mem hay time zone có thể được tùy chỉnh theo nhu cầu doanh nghiệp. Patroni chịu trách nhiệm bầu chọn leader và duy trì trạng thái các node. Khi khởi chạy, node đầu tiên trở thành leader và các node còn lại join vào cluster dưới dạng replica.
	Cấu hình HAProxy HAProxy hoạt động như load balancer và điểm truy cập duy nhất cho client. Nó phân tách các kết nối write (Primary) và read (replica), giúp các ứng dụng kết nối mà không cần biết node nào đang là leader. File cấu hình HAProxy (haproxy.cfg) chỉ định IP/port của các node PostgreSQL, max connection (maxconn), và các listener tương ứng. Việc này giúp tăng khả năng chịu tải và đảm bảo kết nối luôn ổn định.
	Kiểm thử và vận hành cluster HA Sau khi cài đặt và cấu hình xong, cluster có thể được kiểm thử bằng cách kết nối PostgreSQL thông qua HAProxy. Port write cho phép tạo database và ghi dữ liệu, port read-only chỉ cho phép đọc. Khi node Primary gặp sự cố, Patroni tự động bầu chọn node mới làm leader, đảm bảo failover liên tục. Node cũ khi khởi động lại không tự động join cluster mà cần thực hiện recovery thủ công hoặc sử dụng script tự động.
	Lưu ý triển khai thực tế - Cluster nên triển khai tối thiểu 3 node để đảm bảo quorum. - Cấu hình các tham số PostgreSQL phụ thuộc vào tài nguyên server và nhu cầu ứng dụng. - Thư mục lưu mật khẩu PostgreSQL nên được bảo mật và không lưu mặc định. - Để mở rộng cluster, có thể thêm node mới vào Etcd và Patroni từ từ mà không làm gián đoạn cluster hiện tại. - Đọc kỹ tài liệu chính thức Patroni, Etcd, HAProxy để hiểu ý nghĩa các tham số và tối ưu hóa cluster.
Kafka	Triển khai Kafka Cluster với tính khả dụng cao (High Availability – HA) yêu cầu chuẩn bị một môi trường nhiều node để đảm bảo cluster có khả năng chịu lỗi và đồng bộ dữ liệu giữa các broker. Thông thường, một cluster HA cần ít nhất ba server, mỗi server đóng vai trò là một broker trong Kafka. Trước khi cài đặt Kafka, môi trường cần được chuẩn bị đầy đủ với Java Runtime Environment (JRE), vì Kafka được phát triển trên nền tảng Java. Việc cài đặt Java trên tất cả các server là bước cơ bản và cần thiết để các broker có thể chạy ổn định.
	Zookeeper là thành phần quan trọng thứ hai trong kiến trúc Kafka, chịu trách nhiệm quản lý cluster, bầu leader và duy trì đồng bộ metadata giữa các broker. Khi triển khai Zookeeper cluster, mỗi node cần được gán một ID duy nhất và xác định các thông số như dataDir (thư mục lưu trữ dữ liệu), clientPort (mặc định 2181) và server.X=IP:2888:3888 để thiết lập thông tin các node trong cluster. Các port bầu leader và giao tiếp giữa các node cần được mở trong firewall để đảm bảo các node có thể trao đổi dữ liệu. Sau khi cấu hình, cluster Zookeeper cần được khởi động và kiểm tra bằng zkCli.sh bằng cách tạo các znode thử nghiệm để đảm bảo dữ liệu đồng bộ trên tất cả các node.
	Sau khi Zookeeper đã sẵn sàng, Kafka broker có thể được cài đặt. Phiên bản Kafka nên được lựa chọn dựa trên yêu cầu dự án, không nhất thiết phải là phiên bản mới nhất để tránh xung đột với các công cụ khác. Sau khi download và giải nén Kafka, cấu hình server.properties cần được chỉnh sửa cho mỗi broker. Các thông số quan trọng gồm broker.id (ID riêng cho từng node), listeners (địa chỉ IP của broker), log.dirs (thư mục lưu dữ liệu Kafka) và zookeeper.connect (danh sách các node Zookeeper). Việc cấu hình chính xác các tham số này đảm bảo Kafka cluster hoạt động ổn định và đồng bộ với Zookeeper.
	Khởi chạy Kafka cluster có thể thực hiện bằng script kafka-server-start.sh với file cấu hình đã chuẩn bị. Để đảm bảo hệ thống hoạt động HA, cần tạo một topic thử nghiệm với replication-factor lớn hơn 1 để dữ liệu được sao lưu giữa các broker. Sau đó, kiểm tra topic trên tất cả các broker để đảm bảo dữ liệu đồng bộ. Khi cluster hoạt động ổn định, Kafka có thể được quản lý như một systemd service, cho phép tự động start khi hệ thống boot và dễ dàng quản lý các process. Việc tạo một user riêng cho Kafka giúp nâng cao bảo mật và quản lý cluster chuyên nghiệp hơn.
	Tư duy triển khai Kafka HA tập trung vào thực hành và kiểm soát chính xác các thông số cluster. Các yếu tố quan trọng gồm ID của broker, port giao tiếp, replication factor và thư mục lưu trữ dữ liệu. Người mới nên tuân theo hướng dẫn trực tiếp, trong khi những người có kinh nghiệm có thể tùy chỉnh các tham số nâng cao. Bằng cách kết hợp Java, Zookeeper và Kafka broker, một cluster Kafka có thể đảm bảo khả năng chịu lỗi, đồng bộ dữ liệu và sẵn sàng phục vụ các ứng dụng streaming một cách ổn định.
Elasticsearch	Elasticsearch là một công cụ tìm kiếm và phân tích mạnh mẽ, thường được triển khai trong các môi trường doanh nghiệp để đảm bảo khả năng mở rộng và độ sẵn sàng cao. Một cluster Elasticsearch HA thường được xây dựng từ nhiều node, trong đó có các node Master và node Data. Mô hình phổ biến là 3 node: Node1, Node2, Node3. Trong đó, Node Master chính chịu trách nhiệm quản lý cluster và phân phối dữ liệu, còn các node còn lại sẽ tham gia lưu trữ dữ liệu và có thể được đề cử làm Master dự phòng khi node chính gặp sự cố. Kiến trúc này đảm bảo rằng khi một node gặp lỗi, cluster vẫn duy trì trạng thái hoạt động ổn định mà không làm gián đoạn dịch vụ.
	Trước khi triển khai, cần chuẩn bị sẵn các server và cài đặt hệ điều hành, đảm bảo truy cập mạng giữa các node và mở các port cần thiết (9200 cho HTTP, 9300 cho giao tiếp giữa các node). Việc cài đặt Elasticsearch được thực hiện trên từng node, ưu tiên sử dụng phiên bản mới nhất (8.x trở lên) để tận dụng các cải tiến về bảo mật và đồng bộ hóa, đồng thời giảm bớt các bước cấu hình thủ công so với các phiên bản cũ. Các file cấu hình chính của Elasticsearch nằm trong /etc/elasticsearch, trong đó quan trọng nhất là elasticsearch.yml để cấu hình cluster name, node name, network host, port, node Master, và các tùy chọn bảo mật.

	Cluster HA được thiết lập thông qua việc khởi tạo node Master trước và thêm các node còn lại vào cluster bằng token do node Master tạo ra. Token này giúp các node Data xác thực và tham gia cluster một cách an toàn. Sau khi cấu hình, trạng thái cluster và node được kiểm tra thông qua các API của Elasticsearch, đảm bảo rằng các node đều hoạt động và node Master có thể failover khi cần thiết. Trong trường hợp node Master hiện tại bị lỗi, một node khác sẽ tự động được đề cử làm Master mà không ảnh hưởng đến khả năng truy cập dữ liệu hoặc vận hành của cluster.
	Bảo mật là một yếu tố quan trọng trong triển khai Elasticsearch HA. HTTPS được khuyến nghị sử dụng để bảo vệ dữ liệu truyền tải và tránh các lỗi liên quan đến token khi join cluster. Người triển khai cần chuẩn bị các file certificate gồm domain.key, domain.crt, và http_ca.crt, và cấu hình các file này trong cả Elasticsearch và Kibana. Kibana được cài đặt tương tự Elasticsearch và kết nối với cluster thông qua token. Việc này cho phép quản trị viên truy cập, giám sát, và quản lý dữ liệu trong cluster một cách an toàn thông qua giao diện web.
	Sau khi cluster hoạt động ổn định, việc quản lý dữ liệu được thực hiện bằng cách tạo index, đặt số lượng replica phù hợp và post các document vào cluster. Số lượng replica nên được cấu hình để dữ liệu được phân phối trên các node, đảm bảo tính sẵn sàng và khả năng chịu lỗi. Ngoài ra, các file cấu hình nâng cao như jvm.options hay các tùy chỉnh giới hạn tài nguyên có thể được điều chỉnh tùy thuộc vào yêu cầu dự án và hạ tầng doanh nghiệp, giúp tối ưu hiệu năng và độ ổn định của cluster.
	Tóm lại, triển khai một Elasticsearch Cluster High Availability đòi hỏi sự chuẩn bị kỹ lưỡng về hạ tầng, cài đặt chính xác trên từng node, cấu hình cluster và bảo mật, cũng như kiểm tra failover và replication. Khi được triển khai đúng cách, cluster không chỉ đảm bảo dữ liệu được lưu trữ an toàn mà còn duy trì khả năng phục vụ dịch vụ liên tục ngay cả khi một hoặc nhiều node gặp sự cố, đáp ứng các yêu cầu khắt khe trong môi trường doanh nghiệp.
MongoDB	Giới thiệu về MongoDB Cluster và High Availability MongoDB Cluster là một hệ thống cơ sở dữ liệu phân tán gồm nhiều node, trong đó một node được bầu làm Primary (Master) để thực hiện các thao tác đọc và ghi, còn các node còn lại là Secondary (Slave), chỉ phục vụ việc đọc dữ liệu. Mục tiêu chính của cluster là đảm bảo tính khả dụng cao (High Availability – HA) và khả năng chịu lỗi: khi node Primary gặp sự cố, hệ thống tự động bầu chọn một node Secondary làm Primary mới, đảm bảo dữ liệu vẫn được truy xuất và ghi một cách liên tục. Việc triển khai MongoDB Cluster HA giúp doanh nghiệp duy trì hoạt động của các ứng dụng quan trọng mà không bị gián đoạn do lỗi phần cứng hoặc phần mềm.
	Chuẩn bị và cài đặt MongoDB trên nhiều server Để triển khai cluster, cần chuẩn bị ít nhất ba server với kết nối mạng riêng tư giữa chúng. Trên mỗi server, MongoDB cần được cài đặt, khởi động và bật tự động cùng hệ thống. Các công cụ hỗ trợ như net-tools và telnet cũng nên được cài đặt để kiểm tra port và kết nối mạng. MongoDB mặc định chỉ lắng nghe trên localhost, vì vậy file cấu hình /etc/mongod.conf cần chỉnh sửa để bind tất cả các IP trong mạng riêng, đồng thời thiết lập tên cluster thông qua replication.replSetName. Sau khi cấu hình, dịch vụ MongoDB cần được restart để áp dụng.
	Khởi tạo và cấu hình MongoDB Cluster Sau khi cài đặt MongoDB trên cả ba server, cluster được khởi tạo thông qua shell MongoDB (mongo --shell). Cấu hình khởi tạo bao gồm tên cluster và danh sách các node với địa chỉ IP và port. Sau khi khởi tạo, Primary được tự động bầu chọn, các Secondary đồng bộ dữ liệu từ Primary. Trạng thái cluster có thể kiểm tra bằng lệnh rs.status(). Tại đây, dữ liệu được tạo trên Primary sẽ tự động đồng bộ sang các Secondary, nhưng ghi dữ liệu trực tiếp trên Secondary sẽ bị từ chối để đảm bảo tính nhất quán.
	Thử nghiệm tính khả dụng cao (High Availability) Cluster MongoDB HA đảm bảo rằng khi Primary bị tắt hoặc mất kết nối, hệ thống tự động bầu chọn node Secondary có độ ưu tiên cao nhất làm Primary mới. Sau khi node cũ khởi động lại, nó trở thành Secondary và đồng bộ lại dữ liệu. Việc bật tùy chọn enable cho MongoDB trên hệ thống đảm bảo các node khởi động tự động cùng server, duy trì tính ổn định và khả năng phục hồi của cluster.
	Nâng cao: Sử dụng Virtual IP (VIP) Để kết nối đến cluster dễ dàng và chuyên nghiệp, Virtual IP (VIP) được triển khai thông qua Keepalived trên cả ba server. VIP đảm bảo rằng ứng dụng chỉ cần kết nối tới một địa chỉ IP duy nhất, hệ thống tự động điều phối traffic đến node Primary hiện tại. Cấu hình Keepalived bao gồm định nghĩa trạng thái server (MASTER cho node Primary, BACKUP cho các Secondary), interface mạng, Virtual Router ID (VRID) duy nhất trong mạng, priority để xác định node ưu tiên làm Primary, và địa chỉ IP ảo. Khi node Master gặp sự cố, VIP sẽ chuyển sang node Secondary được đề cử dựa trên priority, đảm bảo kết nối vẫn hướng tới node Primary hợp lệ.
	Thiết lập ưu tiên trong MongoDB để đồng bộ với VIP Để VIP luôn gắn với node Primary đúng, cần thiết lập priority cho các node trong MongoDB. Node có priority cao nhất sẽ được bầu làm Primary khi cluster khởi động hoặc khi có sự cố xảy ra. Việc này được thực hiện bằng lệnh rs.conf() và rs.reconfig(), chỉnh sửa thuộc tính members[n].priority của từng node. Kết hợp với VIP, cơ chế này đảm bảo rằng dù ứng dụng kết nối tới một IP duy nhất, dữ liệu luôn được đọc và ghi trên node Primary thực tế, duy trì HA và tính nhất quán dữ liệu.
	Kết luận Việc triển khai MongoDB Cluster với High Availability và Virtual IP là một giải pháp thực tế, giúp doanh nghiệp đảm bảo hoạt động liên tục của ứng dụng, giảm thiểu downtime và đơn giản hóa cấu hình kết nối. Cluster này tự động quản lý việc bầu chọn Primary, đồng bộ dữ liệu giữa các node và sử dụng VIP để cung cấp địa chỉ IP duy nhất cho ứng dụng. Giải pháp này vừa đáp ứng yêu cầu HA, vừa thuận tiện cho việc quản lý và triển khai trong môi trường doanh nghiệp.

MariaDB	Giới thiệu về MariaDB Galera Cluster và High Availability MariaDB Galera Cluster là một giải pháp cơ sở dữ liệu phân tán giúp đảm bảo tính High Availability (HA) và đồng bộ dữ liệu đa node. Khi triển khai cluster, tất cả các server trong cluster đều có dữ liệu giống nhau và có khả năng đồng bộ ngay lập tức, nhờ cơ chế synchronous replication theo hàng (row-based replication). Điều này giúp hệ thống có khả năng chịu lỗi cao, khi một node gặp sự cố, các node còn lại vẫn tiếp tục phục vụ dữ liệu, đảm bảo hoạt động liên tục cho doanh nghiệp.
	Chuẩn bị môi trường và cài đặt MariaDB Trước khi triển khai Galera Cluster, cần chuẩn bị tối thiểu ba server, cài đặt MariaDB trên tất cả các node bằng các gói có sẵn trên hệ điều hành. Việc cài đặt có thể thực hiện bằng lệnh apt install mariadb-server mariadb-client trên Ubuntu hoặc tương tự trên các bản Linux khác. Người dùng nên tạo một user với quyền sudo nếu không muốn sử dụng trực tiếp root. Ngoài ra, cần kiểm tra kết nối mạng giữa các server để đảm bảo cluster hoạt động ổn định.
	Cấu hình Galera Cluster trên từng node Cấu hình Galera Cluster được thực hiện qua các file trong /etc/mysql/conf.d/, thường tạo file riêng như galera.cnf. Các cấu hình quan trọng bao gồm: - binlog_format = row để đảm bảo log ghi theo hàng, phục vụ đồng bộ dữ liệu chính xác. - default_storage_engine = InnoDB hỗ trợ giao dịch và khóa dòng. - innodb_autoinc_lock_mode = 2 cải thiện hiệu suất trong môi trường phân tán. - bind-address = 0.0.0.0 để node có thể kết nối từ mọi IP. - Kích hoạt Galera Cluster (wsrep_on = ON) và chỉ định thư viện Galera (wsrep_provider), tên cluster (wsrep_cluster_name) và danh sách các node (wsrep_cluster_address). Sau khi cấu hình xong, node đầu tiên (Server 1) sẽ được khởi tạo làm Master ban đầu bằng lệnh galera_new_cluster, các node còn lại chỉ cần start MariaDB sẽ tự động join cluster. Kiểm tra trạng thái cluster bằng lệnh SQL SHOW STATUS LIKE 'wsrep_cluster_size', nếu giá trị bằng số node triển khai, cluster hoạt động chính xác.
	Kiểm tra đồng bộ dữ liệu Để kiểm tra đồng bộ, người dùng có thể tạo database và bảng trên server Master và thêm dữ liệu. Dữ liệu sẽ tự động được đồng bộ sang các node khác, đảm bảo tính nhất quán giữa tất cả các node. Đây là cơ chế chính giúp Galera Cluster duy trì HA và đồng bộ dữ liệu trong thời gian thực.
	MaxScale – Proxy quản lý HA và Load Balancing MariaDB MaxScale là một proxy open-source phát triển bởi MariaDB Corporation, giúp quản lý High Availability, load balancing và failover tự động cho Galera Cluster. MaxScale kết nối đến tất cả node trong cluster, phân chia read/write, đồng thời đề cử Master khi node chính gặp sự cố. Để sử dụng MaxScale, cần cài đặt trên các server, cấu hình file maxscale.cnf với danh sách server, tạo user MaxScale trong MariaDB với quyền MONITOR và REPLICATION CLIENT, mở các port cần thiết như 4006, 4008 trên firewall.
	Kiểm tra failover và hoạt động của MaxScale Sau khi khởi động MaxScale, người dùng có thể liệt kê trạng thái các server trong cluster bằng lệnh maxscale control list servers. Khi server Master gặp sự cố, MaxScale sẽ tự động đề cử node khác làm Master, đảm bảo dịch vụ không bị gián đoạn. Khi node cũ khởi động lại, dữ liệu sẽ được đồng bộ với cluster, duy trì tính toàn vẹn dữ liệu.
	Lưu ý và tư duy triển khai - ấu hình có thể được tùy chỉnh theo yêu cầu doanh nghiệp nhưng cần tuân thủ các quy tắc của Galera. - Luôn backup file cấu hình trước khi chỉnh sửa để dễ khôi phục khi gặp lỗi. - Kiểm tra log và trạng thái cluster thường xuyên. - Có thể sử dụng thêm Virtual IP (VIP) để truy cập cluster ổn định, hỗ trợ load balancing và HA.