ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN HỆ THỐNG THÔNG TIN

BÁO CÁO ĐỒ ÁN

CÁC HỆ THỐNG PHÂN TÁN

Đề tài :

HỆ THỐNG KUBERNETES – K8S VÀ TRIỂN KHAI HORIZONTAL POD AUTOSCALING

GIẢNG VIÊN HƯỚNG DẪN : TS. Thái Lê Vinh

HỌC VIÊN THỰC HIỆN :

20C12007 – Trần Đình Lâm

20C12030 – Huỳnh Lâm Phú Sĩ

KHÓA : K30

TP.HCM, 05 tháng 01 năm 2022

MỤC LỤC

Thông tin chi tiết nhóm

BẢNG PHÂN CÔNG & ĐÁNH GIÁ HOÀN THÀNH CÔNG VIỆC			
Người thực hiện	**Công việc thực hiện**	**Mức độ hoàn thành**	**Đánh giá của nhóm**
20C12007 Trần Đình Lâm	Tìm hiểu kiến trúc hệ thống cơ bản	70%	70%
	Tìm hiểu ứng dụng Kubernetes vào giám sát tự động		
	Chạy thực nghiệm kiến trúc ứng dụng		
	Viết báo cáo tổng hợp		
20C12030 Huỳnh Lâm Phú Sĩ	Tìm hiểu kiến trúc hệ thống cơ bản	70%	70%
	Soạn slide trình bày thử nghiệm		
	Viết báo cáo tổng hợp		

Bảng 1: Thông tin chi tiết và phân công nhóm

Danh sách các hình

1. Tổng quan về Kubernetes

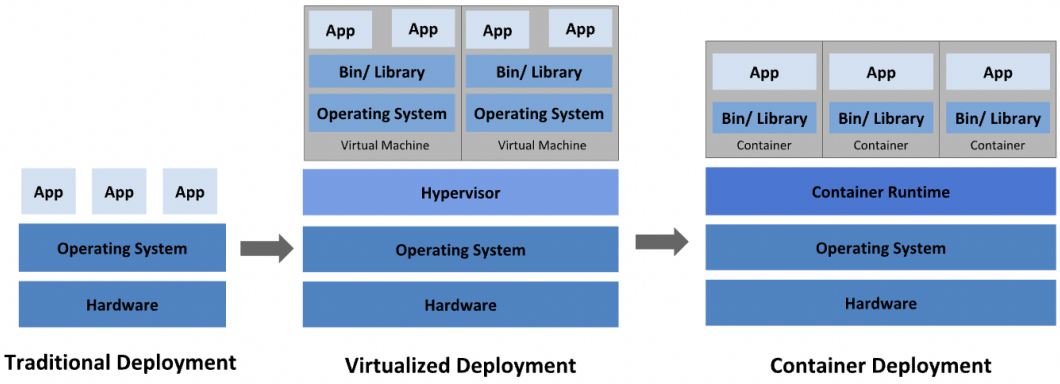
1.1. Đặt vấn đề

Ngày nay, với sự phát triển của internet và nhu cầu thực hiện các tác vụ internet bùng nổ theo từng ngày, các hệ thống phân tán cũng ngày càng phát triển để theo kịp tiến độ phát triển đó. Các hệ thống phân tán ngày nay được phát triển để đáp ứng nhu cầu sử dụng to lớn của người dung cũng như như nhu cầu phát triển của các nhà phát triển. Do đó, nhằm mục đích đơn giản hóa quy trình phát triển và triển khai của các ứng dụng internet, nhiều công cụ mạnh mẽ đã được tạo ra. Trong đó, Kubernetes (K8s) là một hệ thống quản lí mã nguồn mở vô cùng mạnh mẽ được phát triển bởi Google

1.2. Giới thiệu về Kubernetes

Ngày nay, với sự phát triển của internet và nhu cầu thực hiện các tác vụ internet bùng nổ theo từng ngày, các hệ thống phân tán cũng ngày càng phát triển để theo kịp tiến độ phát triển đó. Các hệ thống phân tán ngày nay được phát triển để đáp ứng nhu cầu sử dụng to lớn của người dung cũng như như nhu cầu phát triển của các nhà phát triển. Do đó, nhằm mục đích đơn giản hóa quy trình phát triển và triển khai của các ứng dụng internet, nhiều công cụ mạnh mẽ đã được tạo ra. Trong đó, Kubernetes (K8s) là một hệ thống quản lí mã nguồn mở vô cùng mạnh mẽ được phát triển bởi Google.

1.3. Triển khai ứng dụng bằng container



Hình 1: Quá trình phát triển của kiến trúc hệ thống (Nguồn: kubernetes.io [1])

Cách triển khai theo kiểu truyền thống: Ban đầu, các tổ chức vận hành ứng dụng trên các máy chủ vật lý. Không có cách nào để xác định ranh giới tài nguyên cho các ứng dụng trong máy chủ vật lý và điều này gây ra sự bất hợp lí trong phân bổ tài nguyên. Giải pháp cho điều này là chạy từng ứng dụng trên một máy chủ vật lý khác nhau. Nhưng nó khiến cho việc scale thiếu hiệu quả do tài nguyên không được sử dụng đầy đủ và tốn kém để duy trì nhiều máy chủ vật lý.

Cách triển khai theo kiểu ảo hóa: Như một giải pháp khắc phục các điểm yếu cho cách triển khai truyền thống, ảo hóa đã ra đời. Nó cho phép nhà phát triển chạy nhiều Máy ảo (VM) trên một CPU của một máy chủ vật lý. Ảo hóa cho phép các ứng dụng được tách biệt giữa các máy ảo và cung cấp độ bảo mật cao vì thông tin của một ứng dụng này không thể truy cập tùy tiện bởi ứng dụng khác, giảm chi phí phần cứng và hơn thế nữa.
Cách triển khai bằng container: Các container tương tự như máy ảo, nhưng chúng có đặc tính đóng để chia sẻ Hệ điều hành (OS) giữa các ứng dụng. Vì vậy, container ít chiếm tài nguyên hơn. Tương tự như một máy ảo, một vùng chứa có hệ thống tệp riêng của nó, chia sẻ CPU, bộ nhớ, không gian xử lý và hơn thế nữa. Khi chúng được tách ra khỏi cơ sở hạ tầng bên dưới, chúng có thể di động qua các đám mây và các hệ điều hành khác nhau.

Ưu điểm của triển khai ứng dụng bằng container:

- Tạo và triển khai ứng dụng linh hoạt: tăng tính đơn giản và hiệu quả của việc tạo image container so với việc sử dụng image máy ảo.
- Phát triển, tích hợp và triển khai liên tục (CI/CD): cung cấp giải pháp đáng tin cậy và ổn định cho việc build và deploy image container với khả năng rollback nhanh chóng và hiệu quả (do tính bất biến của image).
- Phân tách Dev và Ops: tạo image container ứng dụng tại thời điểm build / release hơn là thời điểm deploy, do đó tách ứng dụng khỏi cơ sở hạ tầng.
- Khả năng quan sát: không chỉ hiển thị thông tin và số liệu cấp hệ điều hành, mà còn hiển thị tình trạng ứng dụng và các tín hiệu khác.
- Tính nhất quán về môi trường xuyên suốt quá trình develop, test và production: Chạy trên laptop giống như chạy trên đám mây
- Khả năng phân tán trên OS và đám mây: Chạy trên Ubuntu, RHEL, CoreOS, và trên các dịch vụ đám mây công cộng lớn.
- Quản lý tập trung vào ứng dụng: Nâng cao mức độ trừu tượng từ việc chạy một hệ điều hành trên phần cứng ảo sang chạy một ứng dụng trên một hệ điều hành sử dụng tài nguyên logic.
- Các micro-service được kết hợp, phân tán, giải phóng không phụ thuộc lẫn nhau: các ứng dụng được chia thành các phần nhỏ hơn, độc lập và có thể được triển khai và quản lý động.
- Sử dụng tài nguyên hiệu quả cao.

1.4. Kubernetes cung cấp những tính năng gì?

Cách triển khai theo kiểu truyền thống: Ban đầu, các tổ chức vận hành ứng dụng trên các máy chủ vật lý. Không có cách nào để xác định ranh giới tài

nguyên cho các ứng dụng trong máy chủ vật lý và điều này gây ra sự bất hợp lí trong phân bổ tài nguyên. Giải pháp cho điều này là chạy từng ứng dụng trên một máy chủ vật lý khác nhau. Nhưng nó khiến cho việc scale thiếu hiệu quả do tài nguyên không được sử dụng đầy đủ và tốn kém để duy trì nhiều máy chủ vật lý.

Cách triển khai theo kiểu ảo hóa: Như một giải pháp khắc phục các điểm yếu cho cách triển khai truyền thống, ảo hóa đã ra đời. Nó cho phép nhà phát triển chạy nhiều Máy ảo (VM) trên một CPU của một máy chủ vật lý. Ảo hóa cho phép các ứng dụng được tách biệt giữa các máy ảo và cung cấp độ bảo mật cao vì thông tin của một ứng dụng này không thể truy cập tùy tiện bởi ứng dụng khác, giảm chi phí phần cứng và hơn thế nữa.

Cách triển khai bằng container: Các container tương tự như máy ảo, nhưng chúng có đặc tính đóng để chia sẻ Hệ điều hành (OS) giữa các ứng dụng. Vì vậy, container ít chiếm tài nguyên hơn. Tương tự như một máy ảo, một vùng chứa có hệ thống tệp riêng của nó, chia sẻ CPU, bộ nhớ, không gian xử lý và hơn thế nữa. Khi chúng được tách ra khỏi cơ sở hạ tầng bên dưới, chúng có thể di động qua các đám mây và các hệ điều hành khác nhau.

Ưu điểm của triển khai ứng dụng bằng container:

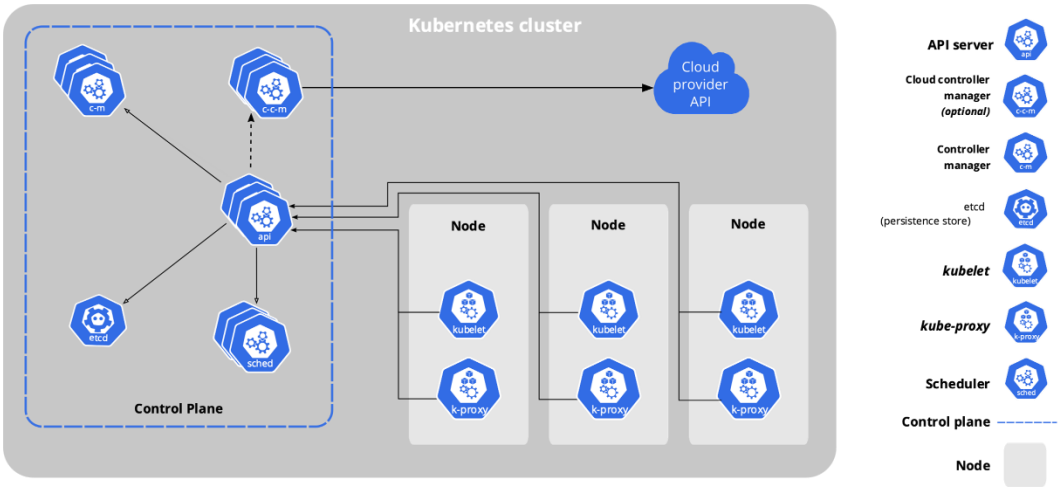
- Tạo và triển khai ứng dụng linh hoạt: tăng tính đơn giản và hiệu quả của việc tạo image container so với việc sử dụng image máy ảo.
- Phát triển, tích hợp và triển khai liên tục (CI/CD): cung cấp giải pháp đáng tin cậy và ổn định cho việc build và deploy image container với khả năng rollback nhanh chóng và hiệu quả (do tính bất biến của image).
- Phân tách Dev và Ops: tạo image container ứng dụng tại thời điểm build / release hơn là thời điểm deploy, do đó tách ứng dụng khỏi cơ sở hạ tầng.
- Khả năng quan sát: không chỉ hiển thị thông tin và số liệu cấp hệ điều hành, mà còn hiển thị tình trạng ứng dụng và các tín hiệu khác.
- Tính nhất quán về môi trường xuyên suốt quá trình develop, test và production: Chạy trên laptop giống như chạy trên đám mây
- Khả năng phân tán trên OS và đám mây: Chạy trên Ubuntu, RHEL, CoreOS, và trên các dịch vụ đám mây công cộng lớn.
- Quản lý tập trung vào ứng dụng: Nâng cao mức độ trừu tượng từ việc chạy một hệ điều hành trên phần cứng ảo sang chạy một ứng dụng trên một hệ điều hành sử dụng tài nguyên logic.

- Các micro-service được kết hợp, phân tán, giải phóng không phụ thuộc lẫn nhau: các ứng dụng được chia thành các phần nhỏ hơn, độc lập và có thể được triển khai và quản lý động.
- Sử dụng tài nguyên hiệu quả cao.

2. Kiến trúc của Kubernetes

2.1. Các thành phần (component) của Kubernetes

Khi triển khai bằng Kubernetes, nhà phát triển sẽ nhận được một Cluster. Kubernetes Cluster bao gồm một tập hợp các Worker Machine, được gọi là các Node, chạy các ứng dụng được chứa trong Container. Mỗi Cluster đều có ít nhất một Worker Node. (Các) Worker Node làm host chứa các Pod là Component của Application Workload. Control Plane quản lý các Worker node và các Pod trong Cluster. Trong môi trường Production, Control Plane thường chạy trên nhiều máy tính và một Cluster thường chạy nhiều Nod, cung cấp khả năng chịu lỗi và tính sẵn sàng cao.



Hình 2: Kiến trúc tổng thể của một K8s cluster (Nguồn:Kubernetes [1])

2.1.1. Các thành phần của Control Plane (Control Plane Components)

Các component của Control Plane đưa ra quyết định chung về Cluster (như lập lịch), cũng như phát hiện và phản hồi các sự kiện của Cluster. Các component của Control Plane có thể chạy trên bất kỳ máy nào trong Cluster. Tuy nhiên, để đơn giản, setup scripts thường khởi động tất cả các Control Plane Component trên cùng một máy và không chạy Container của người người dùng trên máy này.

- kube-apiserver: API Server là một Component Kubernetes Control Plane để public ra các API Kubernetes. API Server giống như front end của Kubernetes Control Plane
- etcd: Là một Key-value store nhất quán và sẵn sàng cao được sử dụng làm "xương sống" cho tất cả dữ liệu Cluster. Nếu Kubernetes Cluster sử dụng etcd để lưu trữ thì nên có một nơi backup các dữ liệu này.
- kube-scheduler: Component thuộc Control Plane theo dõi các Pod mới được tạo mà không có Node được gán và chọn một Node cho Pod đó để chạy dựa trên các yếu tốbao gồm: yêu cầu tài nguyên của riêng nó và tổng thể hệ thống, các ràng buộc về phần cứng / phần mềm / chính sách, thông số kỹ thuật chung,…
- kube-controller-manager: Component của Control Plane chạy các process của controller. Về mặt lý thuyết, mỗi controller là một process riêng biệt, nhưng để giảm độ phức tạp, tất cả chúng đều được biên dịch thành một tệp nhị phân duy nhất và chạy trong một process duy nhất. Có nhiều loại controller-manager như: Node controller, Job controller, Endpoint controller, Service Account & Token controller,…
- cloud-controller-manager: Component của Control Plane được chứa logic điều khiển dành riêng cho đám mây. Cloud Controller Manager cho phép liên kết cluster của nhà phát triển với API của nhà cung cấp dịch vụ đám mây và tách các thành phần tương tác với nền tảng đám mây đó khỏi các thành phần chỉ tương tác với cluster của nhà phát triển. Tương tự như với kube-controller-manager, cloud-controller-manager kết hợp tiến trình khiển độc lập về mặt logic thành một tệp nhị phân duy nhất mà chạy như một quy trình duy nhất.

2.1.2. Các thành phần của Node (Node Components)

Các Node Component chạy trên mọi Node, duy trì các cluster đang chạy và cung cấp cho chúng Kubernetes Runtime Environment.

- Kubelet: Một Agent chạy trên mỗi node trong cluster. Nó đảm bảo rằng các container đang chạy trong Pod. Kubelet lấy một tập hợp các PodSpec được cung cấp thông qua các cơ chế khác nhau và đảm bảo rằng các container được mô tả trong các PodSpec đó đang chạy và hoạt động tốt. Kubelet không quản lý các container không được tạo bởi Kubernetes.
- Kube-proxy: kube-proxy là network proxy chạy trên mỗi node trong cluster, triển khai một phần của khái niệm Kubernetes Service. kube-proxy duy trì các quy tắc mạng trên các nút. Các quy tắc mạng này cho

phép giao tiếp với Pod từ các phiên mạng bên trong hoặc bên ngoài cluster.

- Container runtime: Container runtime là phần mềm chịu trách nhiệm chạy các container.

Ngoài các thành phần kể trên, trong kiến trúc của Kubernetes còn có các Add-on sử dụng tài nguyên của Kubernetes như (Daemon set, deployment,…) để thực hiện một số tính năng của Cluster. Một số Add-on phổ biến được sử dụng như DNS, WebUI (dashboard), Container Resource Monitoring, Cluster-level logging,…

2.2. Các công cụ được sử dụng để tương tác với node Kubernetes

Các nhà phát triển, vận hành hệ thống không thao tác trực tiếp với các thành phần trên của Kubernetes mà thao tác qua các thành phần khác được Kubernetes quy định để giúp người dùng vận hành hệ thống đơn giản và hiệu quả hơn. Chúng bao gồm:

- Pod: là đơn vị nhỏ nhất trong kiến trúc Kubernetes, nó là lớp trừu tượng bao lại các container. Khi triển khai ứng dụng bằng Kubernetes thì một pod chỉ nên thực hiện 1 application. Mỗi pod có một địa chỉ IP của riêng nó và tồn tại với nó trong suốt thời gian nó tồn tại, các thành phần khác trong hệ thống có thể giao tiếp với nó thông qua địa chỉ IP này. Do đó, trong hệ thống khi một pod bị kill do không đáp ứng nhu cầu, các thành phần khác cần biết được IP của pod mới thay thế nó gây ra khá nhiều bất tiện trong quản lí.
- Service: là một địa chỉ IP vĩnh viễn và được dùng thay thế cho IP của Pod. Life cycle của service và pod không phụ thuộc lẫn nhau. Khi một pod bị kill thì service vẫn còn tồn tại và pod mới được dùng để thay thế pod cũng có cùng service với pod đã bị kill. Service gồm có 2 loại là internal service và external service
- Ingress: là phương tiện để forward service ra bên ngoài
- Config map: config map là tập hợp các thiết lập (configuration của application hay pod) được đặt ở bên ngoài. Config map được sử dụng nhằm giải quyết tình trạng thay đổi config liên tục ở các pod khiến ứng dụng phải được build đi build lại nhiều lần. Bằng cách thay đổi config map, nhà phát triển có thể dễ dàng thay đổi thiết lập của ứng dụng mà không cần phải build ứng dụng lại từ đầu.
- Secrect: Một số thiết lập cho ứng dụng hay pod có thể chứa thông tin nhạy cảm (password, key, credential,…) của ứng dụng hoặc của người dùng quản trị của ứng dụng đó. Khi đó secrect được sử dụng để lưu các thiết lập này thay cho config map.

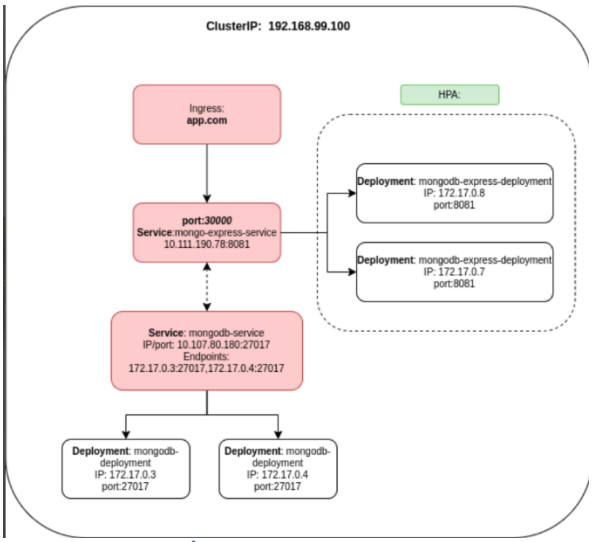
- Volume: là một thành phần lưu trữ dùng để lưu trữ dữ liệu cục bộ hoặc từ xa. Nó không trực tiếp lưu trữ hay chứa thông tin hoặc dữ liệu mà giống như một external drive được cắm vào pod khi có yêu cầu về lưu trữ.
- Deployment: deployment được xem như một bản thiết kết của pod. Đây là nơi mà người dùng sẽ thiết lập cho pod, sau đó hệ thống sẽ sử dụng deployment để triển khai pod, do đó người dùng sẽ không tương tác trực tiếp với pod. Dựa trên deployment mà hệ thống sẽ scale up/down các pod. Tuy nhiên chỉ sử dụng Deployment cho stateless application vì các dữ liệu được tạo ra trong quá trình chạy app không bền vững và sẽ bị mất đi trong quá trình scale up/down các replica của pod.
- Statefullset: thành phần lưu trữ nhất quán của các ứng dụng trong Kubernetes. Triển khai các statefullset không dễ dàng vì có thể gây ra sự không nhất quán về dữ liệu trong quá trình scale. Ngoài ra, người ta thường triển khai các DB cho các statefullset bên ngoài Cluster nhằm tăng tính nhất quán và đảm bảo an toàn cho dữ liệu trong quá trình hoạt động và scale.
- Replica set: quản lí tập các replica của pod, chứa các thông tin về quá trình triển khai, hoạt động và scale của các pod cùng loại.

3. Triển khai ứng dụng Horizontal Pod Autoscaler (HPA)

3.1. Triển khai HPA bằng Kubernetes

Trong K8s, Autoscaler là các thành phần liên quan đến hoạt động tăng giảm tự động số lượng các Pod bên trong một node, hoặc giữa nhiều node với nhau nhằm phục vụ lượng tải của ứng dụng tại một thời điểm nhất định. HPA thường được sử dụng cho các ứng dụng dạng stateless vì tính đơn giản của nó.

Cơ chế hoạt động của HPA dựa vào cấu hình ban đầu cho một hằng số tài



Hình 3: Một kiến trúc ứng dụng có sử dụng HPA

nguyên như CPU, RAM hoặc network. Các Pod sẽ gửi thông số tài nguyên đến

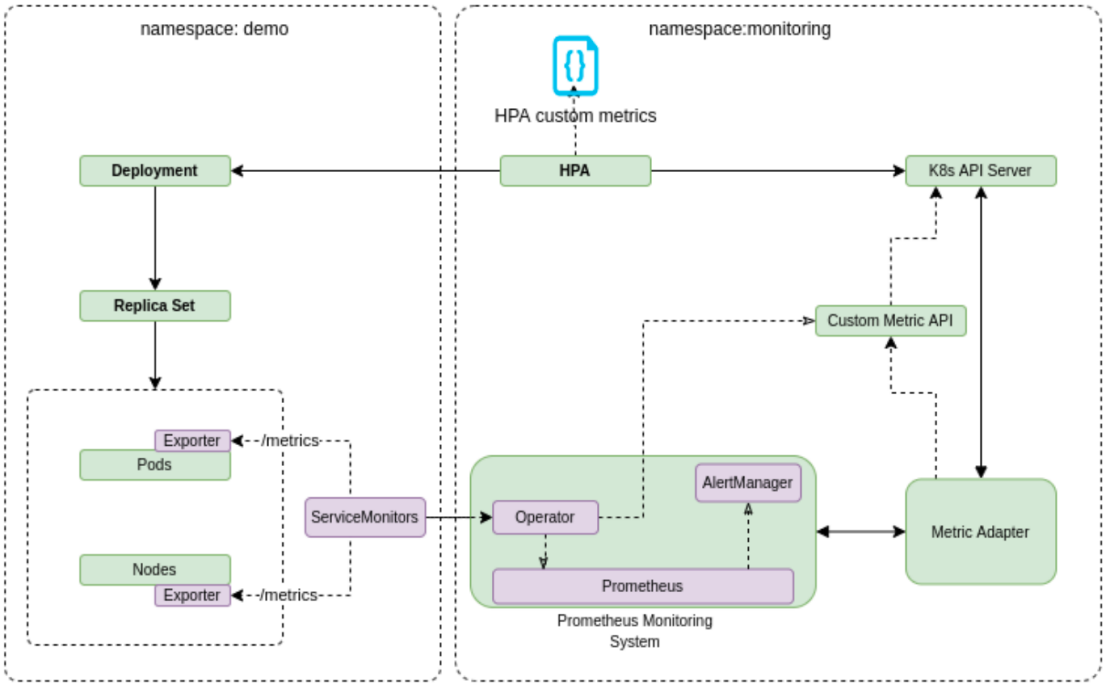
một metric-serer mỗi 10 giây. Dựa vào đó, Autoscaler sẽ giám sát các thông số đó từ metric-server, và lấy giá trị trung bình *m* của tài nguyên đó. Nếu *m* vượt ngưỡng cho trước thì tăng thêm số Pod sao cho *m* về lại được giá trị mong đợi. Điều tương tự cũng được áp dụng cho trường hợp giảm số Pod.

Ngoài các thông số mặc định có thể dễ dàng thu thập và triển khai đã liệt kê, HPA còn giám sát được các loại điều kiện và thông số phức tạp khác mà người quản trị có thể tùy ý cấu hình và lựa chon cho phù hợp với nhu cầu của ứng dụng. Kiến trúc triển khai ở phần sau cũng sẽ sử dụng cơ chế này để minh họa cho một trường hợp thực tiễn.

3.2. Giới thiệu về Prometheus

Prometheus là một time series database chuyên dùng để thu thập và lưu trữ các thông số của một hay nhiều hệ thống và ứng dụng nào đó, nhằm mục đích trực quan hóa và giám sát sức khỏe, tình trạng của ứng dụng để đưa ra những hành động phù hợp và kịp thời. Trong Kubernetes, Prometheus thường được tích hợp cùng nhiều thành phần khác để phục vụ giám sát hiệu quả và dễ mở rộng hơn, sẽ được trình bày ở kiến trúc triển khai bên dưới.

3.3. Kiến trúc triển khai hệ thống HPA sử dụng custom metrics [1]



Hình 4: Kiến trúc triển khai HPA sử dụng custom metrics

Kiến trúc hệ thống bao gồm 2 namespace “demo” (chuyên trách chạy các ứng dụng) và “monitoring” (giám sát hệ thống và thực hiện các thao tác autoscaling theo metrics đã lựa chọn). Để Prometheus hoạt động được ta cần khởi tạo một Operator đính kèm, sau đó đăng ký một ServiceMonitor để nhận các dữ liệu

thông số từ ứng dụng phía namespace "demo". Ứng dụng demo bao gồm các Deployment, từ đó tạo dựng ra các Pod cụ thể chạy thật dựa trên các Deployment. Ứng với mỗi Pod của ứng dụng, ta đăng ký một đường dẫn /metrics tới ServiceMonitor để đẩy sang các thành phần giám sát.

```
# HELP http_requests_total Total number of http requests
# TYPE http_requests_total counter
http_requests_total{method="POST"} 8706
```

Hình 5: Mẫu custom metric từ Pod ứng dụng, theo chuẩn của Prometheus

Sau khi chạy ứng dụng và cung cấp được metric, ta cần khởi tạo một Metric Adapter để lấy thông số metric như CPU, RAM và các custom metric tự định nghĩa, nhằm mục đích cung cấp các điều kiện để HPA có thể hiểu được, chẳng hạn như thông số "http_requests_total".

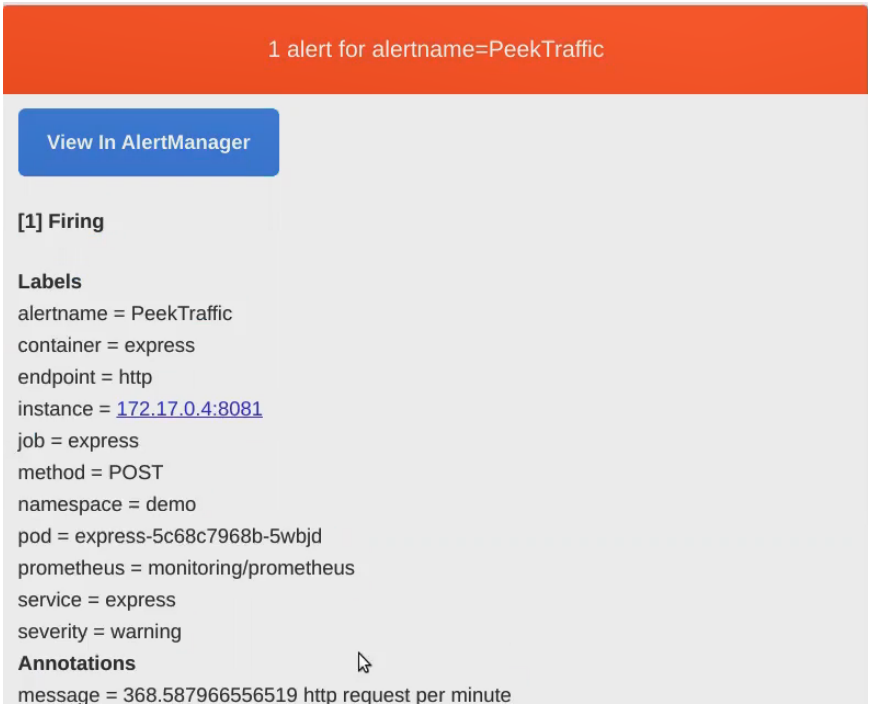
Bên cạnh đó, để thông báo cho người quản trị biết được tình trạng của hệ thống, cũng như sự thay đổi thông số khi có sự kiện scaling diễn ra, ta sử dụng thêm AlertManager tích hợp chung với Prometheus, đọc dữ liệu từ Prometheus và dựa theo điều kiện định sẵn, chẳng hạn số http_request_per_minutes. Khi có traffic đổ vào ứng dụng thì AlertManager gửi thông báo qua tin nhắn hoặc email đến người quản trị.

3.4. Thực nghiệm

Dựa theo kiến trúc triển khai HPA như trên, nhóm đã tiến hành cài đặt và chạy thử nghiệm trên một ứng dụng đơn giản, sau đó giả lập lượng traffic ảo đổ vào ứng dụng.

Hình 6: Quá trình Autoscale theo custom metrics

Service HPA dùng để điều chỉnh Autoscale ban đầu được chỉ định tối đa 10 bản sao. Khi thông số vượt ngưỡng an toàn, quá trình scale được kích hoạt, đồng thời gửi thông báo cảnh báo đến cho người quản trị.



Hình 7: Mẫu thông báo cảnh báo đến người quản trị

3.5. Tổng kết và nhận xét

Sau quá trình cài đặt và thử nghiệm, nhóm rút ra các kết quả đạt được sau đây:

- Tìm hiểu và triển khai ứng dụng chạy trên môi trường container bằng Kubernetes
- Cài đặt, thực hiện HPA trên các pod sử dụng custom metrics và monitor bằng Prometheus có notification bằng Email
- Đánh giá ưu điểm nhược điểm của Kubernetes thông qua thực nghiệm và so sánh với phương pháp triển khai truyền thống

Về ưu nhược điểm của hệ thống vận hành theo kiến trúc đã trình bày, ta có thể liệt kê được các nhận xét sau:

Ưu điểm:

- Triển khai nhanh, đơn giản thông qua hệ sinh thái khổng lồ của Kubernetes
- Kubernetes giúp cho ứng dụng chạy ổn định hơn, hạn chế downtime của hêh thống và dễ dàng thực hiện backup/rollback/rollout.

- Kubernetes giúp giảm chi phí triển khai ứng dụng (máy móc, nhân sự, quy trình,...)
- Miễn phí, mã nguồn mở và cộng đồng phát triển mạnh

Nhược điểm:

- Kubernetes là một hệ thống phức tạp, nó có thể giúp tăng hiệu suất triển khai cũng như hoạt động của ứng dụng nếu mọi thứ trơn tru, ngược lại sẽ rất khó debug khi xảy ra sự cố. Mặt khác, triển khai bằng Kubernetes khá khó khăn đối với những người chưa có kinh nghiệm Devops
- Quá trình chuyển đổi sang triển khai bằng Kubernetes có thể rất phức tạp do nhiều nguyên nhân: ứng dụng chưa được container hóa, ước lượng tài nguyên dành cho Kubernetes trong quá trình vận hành,...
- Nếu việc triển khai không thuận lợi thì việc sử dụng Kubernetes có thể tốn kém hơn cả phương pháp thông thường do quá trình triển khai chậm và thiếu hiệu quả

Tài liệu tham khảo

[1] T. K. Authors, "Kubernetes," [Online]. Available: https://kubernetes.io/.

[2] Anton, "Kubernetest Tutorial," [Online]. Available: https://github.com/antonputra/tutorials.