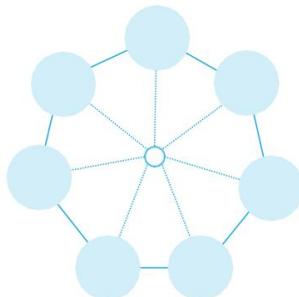


MUMSHAD MANNAMBETH



# kubernetes

cho những người mới bắt đầu

mumshad mannambeth

Xin chào và chào mừng bạn đến với khóa học này trên Kubernetes dành cho người mới bắt đầu. Tôi là Mumshad Mannambeth và tôi sẽ là người hướng dẫn bạn cho khóa học này.

Tôi là ai?

MUMSHAD MANNAMBETH

I am an IT Solutions Architect focusing on Cloud automation and DevOps. I am passionate about learning new technology and teaching. I believe the best way to learn is to learn by doing and in a fun way. I have authored multiple courses on DevOps and cloud automation technologies. My courses focus on providing students with an interactive and hands-on experience in learning new technology that makes learning really interesting.

Total Students 17,375 Courses 5 Reviews 1,849

Courses you are teaching

| Course Title                                 | Author             | Rating          | Price            |
|--|--------------------|-----------------|------------------|
| Docker - SWARM   SERVICES   STACKS - ...     | Mumshad Mammambeth | ★★★★★ 4.6 (81)  | \$499.99 \$11.99 |
| Docker for the Absolute Beginner - Hands On  | Mumshad Mammambeth | ★★★★★ 4.5 (564) | \$64.99 \$11.99  |
| Ansible Advanced                             | Mumshad Mammambeth | ★★★★★ 4.4 (152) | \$154.99 \$11.99 |
| Ansible for the Absolute Beginner - Hands-On | Mumshad Mammambeth | ★★★★★ 4.5 (991) | \$99.99 \$11.99  |

Về tôi, tôi là Kiến trúc sư giải pháp và tôi làm việc về Tự động hóa đám mây và Thực hành DevOps. Tôi đã viết một số khóa học Bán chạy nhất và Được xếp hạng cao nhất trên Udemy về các chủ đề như Docker và Ansible.

## Cấu trúc khóa học



Bài học



Bản demo



Đồ



Bài tập viết mã



Mẹo &amp; thủ thuật



Phản công



Hỏi đáp

Chúng ta hãy xem cấu trúc của khóa học này. Khóa học này bao gồm một loạt bài giảng về Kubernetes, chúng ta sẽ thảo luận về các khái niệm cơ bản về Kubernetes tại đây. Tiếp theo là một số bản demo về cách thiết lập môi trường thử nghiệm và bắt đầu với Kubernetes.

Chúng tôi có các câu hỏi tùy chọn để kiểm tra kiến thức của bạn sau mỗi bài giảng. Đó là hoàn toàn tùy chọn, bạn không cần phải vượt qua những điều này để tiếp tục khóa học.

Sau đó, chúng tôi có các bài tập viết mã giúp bạn học và thực hành phát triển YAML các tệp sẽ cho phép bạn tự phát triển các tệp cấu hình Kubernetes.

Các tệp cấu hình này có thể hơi phức tạp đối với người mới bắt đầu, vì vậy tôi cũng sẽ cung cấp một số mẹo và thủ thuật giúp bạn ghi nhớ định dạng. Điều này cộng với các bài tập viết mã sẽ giúp bạn có đủ tự tin để phát triển các tệp cấu hình của riêng mình. Trên thực tế, chúng tôi sẽ dành THÊM thời gian cho các tệp cấu hình này, vì để làm việc với Kubernetes, bạn PHẢI hiểu rõ về nó.

Cuối cùng, bạn sẽ được giao một bài tập để kiểm tra kiến thức của mình về chủ đề này. Nhiệm vụ này sẽ giúp bạn có được cơ hội tiếp xúc khi thực hiện một nhiệm vụ thực tế với Kubernetes.

Và như mọi khi, nếu có bất kỳ câu hỏi nào, bạn có thể liên hệ trực tiếp với tôi thông qua phần Hỏi đáp.

## Cái này cho ai?



Nhà phát triển



Quản trị viên hệ thống



Người quản lý

Vậy khóa học này dành cho ai? ĐIỀU NÀY dành cho người mới bắt đầu tuyệt đối. Bạn không cần phải có bất kỳ kinh nghiệm nào trước khi làm việc với Kubernetes. Chúng tôi sẽ hướng dẫn tất cả các khái niệm cơ bản cần thiết để bạn hiểu Kubernetes. Điều này dành cho những người mới làm quen với vùng chứa và công nghệ đi kèm với vùng chứa. Bạn có thể là một nhà phát triển đang cố gắng tìm hiểu các dịch vụ vi mô, bộ nhớ và công nghệ đi kèm - đang mong muốn tích lũy một số kinh nghiệm thực tế trong việc phát triển các giải pháp Kubernetes. Bạn có thể là quản trị viên hệ thống đang tìm cách nâng cao kiến thức của mình về các cụm vùng chứa và đang cố gắng tích lũy kinh nghiệm thực tế trong việc thiết lập các cụm như vậy. Bạn có thể là Người quản lý dự án đang tìm cách chỉ hiểu biết cơ bản về các khái niệm cơ bản. Điều này dành cho bất kỳ người mới bắt đầu hành trình Kubernetes nào.

## Làm thế nào để tham dự khóa học này?



Vì vậy, điều đó đưa chúng ta đến câu hỏi về cách bạn nên tham gia khóa học này. Tùy thuộc vào sở thích của bạn cũng như thời gian và nguồn lực bạn có trong tay, bạn có thể chọn bỏ qua một số phần nhất định của khóa học. Nếu bạn đã quen thuộc với các khái niệm cơ bản như bộ chứa docker và tệp YAML, vui lòng bỏ qua các phần đó.

Trong khóa học này, chúng ta sẽ thấy các cách thiết lập kubernetes khác nhau - trên máy cục bộ của bạn hoặc trên các nền tảng đám mây khác nhau như Google Cloud hoặc Amazon, v.v. Nếu bạn đang vội hoặc không có đủ tài nguyên để thiết lập các phiên bản của riêng mình, chúng tôi cũng sẽ thấy tùy chọn có sẵn để tạo cụm kubernetes chỉ trong vài giây trên đám mây.

Cũng nên nhớ khóa học này là dành cho người mới bắt đầu. Bạn sẽ thấy chúng tôi thảo luận về những nội dung thực sự cơ bản, chúng tôi sẽ tập trung vào từng khái niệm và cố gắng hiểu nó một cách đơn giản. Khi chúng tôi làm điều đó, nếu bạn có nghi ngờ về các lĩnh vực khác, hãy giữ lấy chúng. Bởi vì chúng ta có thể thảo luận chúng trong bài giảng sau. Cũng lưu ý rằng đôi khi bạn có thể nghe thấy tôi lặp lại một số câu hoặc cụm từ nhất định và chúng ta có thể tóm tắt lại những gì đã học trong các bài giảng trước thường xuyên hơn. Vì vậy, hãy thông cảm vì tôi làm điều này để đảm bảo rằng các khái niệm chính sẽ được ghi nhớ trong đầu bạn và việc tóm tắt lại thường xuyên hơn sẽ giúp bạn liên hệ các khái niệm mới hơn với những khái niệm cũ tốt hơn.

# Mục tiêu

- Tổng quan về Kubernetes
- Container - Docker
- Dàn nhạc container?
- Demo - Thiết lập Kubernetes
- Khái niệm Kubernetes - POD | Bản sao | Triển khai | Dịch vụ
- Kết nối mạng trong Kubernetes
- Quản lý Kubernetes - Kubectl
- Tệp định nghĩa Kubernetes- YAML
- Kubernetes trên đám mây - AWS/GCP

Chuyển sang các chủ đề chúng tôi sẽ đề cập. Trong khóa học này, chúng ta sẽ tìm hiểu những kiến thức cơ bản về Kubernetes, có gắng hiểu container là gì và điều phối container là gì. Chúng ta sẽ thấy các cách thiết lập và bắt đầu khác nhau với Kubernetes. Chúng ta sẽ xem xét các khái niệm khác nhau như POD, Bản sao, Triển khai và Dịch vụ. Chúng ta sẽ hiểu những điều cơ bản về Mạng trong kubernetes. Chúng ta cũng sẽ dành chút thời gian làm việc với tiện ích dòng lệnh kubectl và phát triển các tệp định nghĩa YAML của kubernetes. Và cuối cùng chúng ta sẽ xem cách triển khai một ứng dụng microservices trên nền tảng đám mây công cộng như Google Cloud.

Như mọi khi, hãy thoải mái tham gia khóa học này theo tốc độ của riêng bạn. Có thể có những phần trong khóa học mà bạn có thể đã quen thuộc nên hãy bỏ qua chúng. Hãy bắt đầu và tôi sẽ gặp bạn trong học phần đầu tiên.

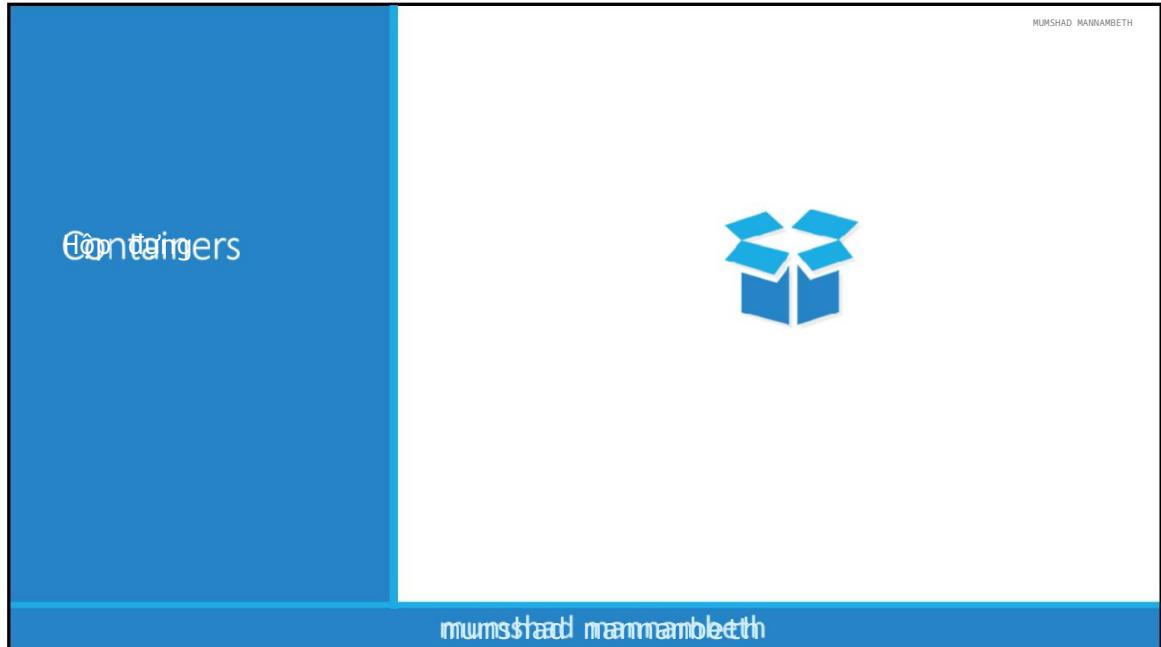
kubernetes hoặc K8

## Vùng chứa + Điều phối

Trong bài giảng này chúng ta sẽ tìm hiểu tổng quan về Kubernetes.

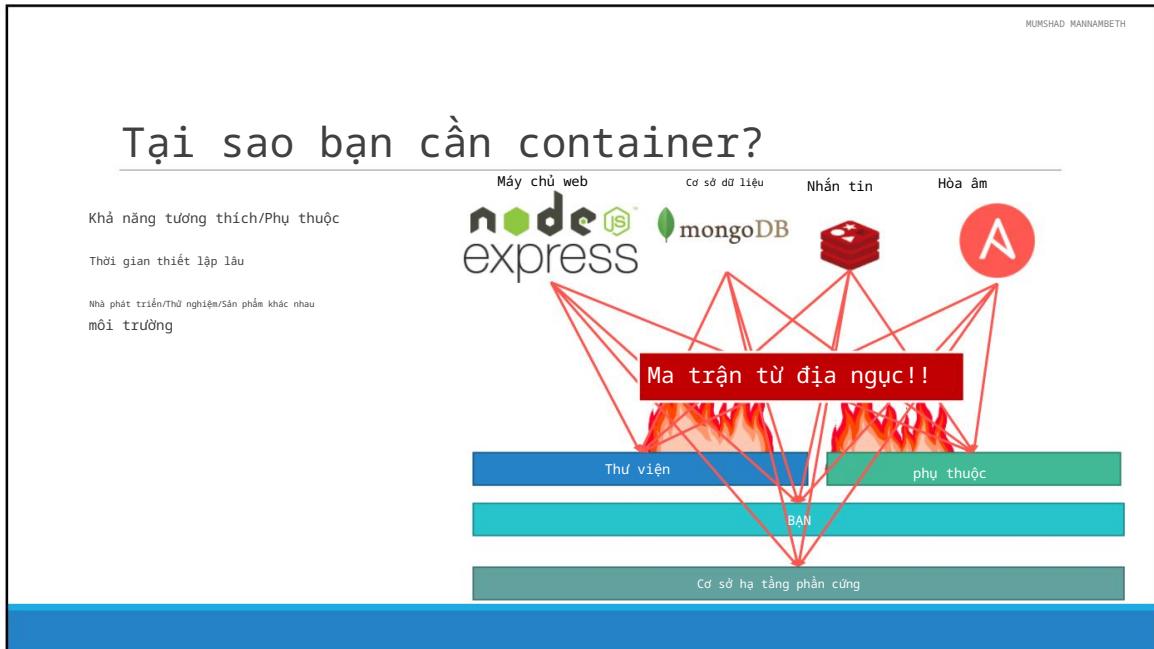
Kubernetes hay còn gọi là K8s được Google xây dựng dựa trên kinh nghiệm chạy container trong sản xuất. Nó hiện là một dự án nguồn mở và được cho là một trong những công nghệ điều phối vùng chứa tốt nhất và phổ biến nhất hiện có. Trong bài giảng này, chúng ta sẽ cố gắng hiểu Kubernetes ở mức độ cao.

Để hiểu Kubernetes, trước tiên chúng ta phải hiểu hai điều - Container và Orchestration. Khi đã làm quen với cả hai thuật ngữ này, chúng ta sẽ có thể hiểu được khả năng của kubernetes. Chúng ta sẽ bắt đầu xem xét từng điều này tiếp theo.



Bây giờ chúng ta sẽ xem container là gì, cụ thể là chúng ta sẽ xem xét công nghệ container phổ biến nhất hiện nay - Docker.

Nếu bạn đã quen thuộc với Docker, vui lòng bỏ qua bài giảng này và chuyển sang bài tiếp theo.



Hãy để tôi bắt đầu bằng việc chia sẻ cách tôi được làm quen với Docker. Trong một trong những dự án trước đây của tôi, tôi đã có yêu cầu thiết lập một ngăn xếp đầu cuối bao gồm nhiều công nghệ khác nhau như Máy chủ Web sử dụng NodeJS và cơ sở dữ liệu như MongoDB/CouchDB, hệ thống nhắn tin như Redis và công cụ điều phối như Ansible. Chúng tôi gặp rất nhiều vấn đề khi phát triển ứng dụng này với tất cả các thành phần khác nhau này. Đầu tiên, khả năng tương thích của chúng với hệ điều hành cơ bản. Chúng tôi phải đảm bảo rằng tất cả các dịch vụ khác nhau này đều tương thích với phiên bản HĐH mà chúng tôi dự định sử dụng. Đôi khi, một số phiên bản nhất định của các dịch vụ này không tương thích với HĐH và chúng tôi phải quay lại và tìm kiếm một HĐH khác tương thích với tất cả các dịch vụ khác nhau này.

Thứ hai, chúng tôi phải kiểm tra tính tương thích giữa các dịch vụ này với các thư viện và phần phụ thuộc trên HĐH. Chúng tôi gặp vấn đề là một dịch vụ yêu cầu một phiên bản của thư viện phụ thuộc trong khi dịch vụ khác yêu cầu một phiên bản khác.

Kiến trúc ứng dụng của chúng tôi thay đổi theo thời gian, chúng tôi phải nâng cấp lên phiên bản mới hơn của các thành phần này hoặc thay đổi cơ sở dữ liệu, v.v. và mỗi khi có điều gì đó thay đổi, chúng tôi phải thực hiện cùng một quy trình kiểm tra tính tương thích giữa các thành phần khác nhau này và cơ sở hạ tầng cơ bản. Cái này

vấn đề ma trận tương thích thường được gọi là ma trận từ địa ngục.

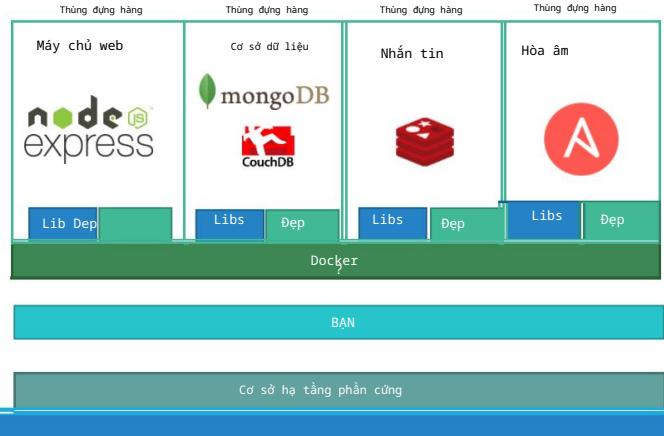
Tiếp theo, mỗi khi có một nhà phát triển mới tham gia, chúng tôi thực sự cảm thấy khó khăn khi thiết lập một môi trường mới. Các nhà phát triển mới phải tuân theo một loạt hướng dẫn lớn và chạy hàng trăm lệnh để cuối cùng thiết lập môi trường của họ. Họ phải đảm bảo rằng họ đang sử dụng đúng Hệ điều hành, đúng phiên bản của từng thành phần này và mỗi nhà phát triển phải tự mình thiết lập tất cả những điều đó.

Chúng tôi cũng có môi trường sản xuất và thử nghiệm phát triển khác nhau. Một nhà phát triển có thể cảm thấy thoải mái khi sử dụng một hệ điều hành và những người khác có thể đang sử dụng một hệ điều hành khác và vì vậy chúng tôi không thể đảm bảo ứng dụng mà chúng tôi đang xây dựng sẽ chạy giống nhau trong các môi trường khác nhau. Và vì vậy tất cả những điều này khiến cuộc sống của chúng tôi trong việc phát triển, xây dựng và vận chuyển ứng dụng thực sự khó khăn.

## Nó có thể làm gì?

Chứa các ứng dụng

Chạy từng dịch vụ với các phần phụ  
thuộc riêng trong các vùng chứa riêng biệt



Vì vậy, tôi cần thứ gì đó có thể giúp chúng tôi giải quyết vấn đề tương thích. Và thứ gì đó sẽ cho phép chúng tôi sửa đổi hoặc thay đổi các thành phần này mà không ảnh hưởng đến các thành phần khác và thậm chí sửa đổi hệ điều hành cơ bản theo yêu cầu. Và tìm kiếm đó đã đưa tôi đến Docker. Với Docker, tôi có thể chạy từng thành phần trong một vùng chứa riêng biệt - với các thư viện riêng và các phần phụ thuộc riêng.

Tất cả đều trên cùng một VM và HĐH, nhưng trong các môi trường hoặc vùng chứa riêng biệt. Chúng tôi chỉ cần xây dựng cấu hình docker một lần và tất cả các nhà phát triển của chúng tôi giờ đây có thể bắt đầu bằng lệnh "docker run" đơn giản. Bất kể họ chạy hệ điều hành cơ bản nào, tất cả những gì họ cần làm là đảm bảo rằng họ đã cài đặt Docker trên hệ thống của mình.

## Container là gì?



Vậy container là gì? Các thùng chứa là môi trường hoàn toàn biệt lập, vì chúng có thể có các quy trình hoặc dịch vụ riêng, giao diện mạng riêng, giá trị gắn kết riêng, giống như Máy ảo, ngoại trừ việc chúng đều có chung một nhân hệ điều hành. Chúng ta sẽ xem xét điều đó có nghĩa gì một chút. Nhưng điều quan trọng cần lưu ý là container không phải là điều mới mẻ với Docker. Container đã tồn tại khoảng 10 năm nay và có một số loại container khác nhau là container LXC, LXD. Việc , LXCFS, v.v. Docker sử dụng LXC thiết lập các môi trường vùng chứa này rất khó vì chúng ở mức rất thấp và đó là Docker cung cấp một công cụ cấp cao với một số chức năng mạnh mẽ giúp người dùng cuối như chúng tôi thực sự dễ dàng.

## Hệ điều hành



Để hiểu cách Docker hoạt động, trước tiên chúng ta hãy xem lại một số khái niệm cơ bản về Hệ điều hành. Nếu bạn nhìn vào các hệ điều hành như Ubuntu, Fedora, Suse hoặc Centos – tất cả chúng đều bao gồm hai thứ. Một hạt nhân hệ điều hành và một bộ phần mềm. Hạt nhân hệ điều hành chịu trách nhiệm tương tác với phần cứng cơ bản. Mặc dù nhân hệ điều hành vẫn giữ nguyên - trong trường hợp này là Linux, nhưng chính phần mềm bên trên nó đã làm cho các Hệ điều hành này trở nên khác biệt. Phần mềm này có thể bao gồm Giao diện người dùng, trình điều khiển, trình biên dịch, Trình quản lý tệp, công cụ dành cho nhà phát triển khác nhau, v.v. VẬY, bạn có một Nhân Linux chung được chia sẻ trên tất cả các Os và một số phần mềm tùy chỉnh phân biệt các hệ điều hành với nhau

## Chia sẻ hạt nhân



Chúng tôi đã nói trước đó rằng các vùng chứa Docker chia sẻ hạt nhân cơ bản. Điều đó thực sự có nghĩa là gì - chia sẻ kernel? Giả sử chúng ta có một hệ thống có hệ điều hành Ubuntu được cài đặt Docker trên đó. Docker có thể chạy bất kỳ phiên bản hệ điều hành nào trên nó miễn là tất cả chúng đều dựa trên cùng một kernel - trong trường hợp này là Linux. Nếu hệ điều hành cơ bản là Ubuntu, docker có thể chạy vùng chứa dựa trên bản phân phối khác như debian, fedora, suse hoặc centos. Mỗi bộ chứa docker chỉ có phần mềm bổ sung mà chúng ta vừa nói đến trong trang trình bày trước, điều này làm cho các hệ điều hành này trở nên khác biệt và docker sử dụng nhân cơ bản của máy chủ Docker hoạt động với tất cả các Os ở trên.

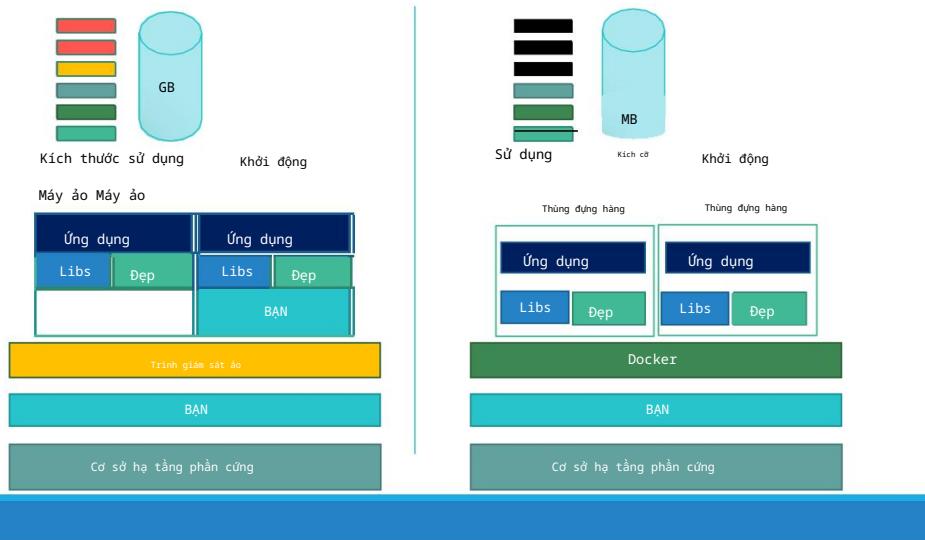
Vậy hệ điều hành nào không dùng chung kernel với những hệ điều hành này? Các cửa sổ ! Và thế là bạn sẽ không thể chạy vùng chứa dựa trên windows trên máy chủ Docker có hệ điều hành Linux trên đó.

Để làm được điều đó, bạn sẽ cần docker trên máy chủ windows.

Bạn có thể hỏi đó không phải là một bất lợi sao? Không thể chạy kernel khác trên hệ điều hành? Câu trả lời là không! Bởi vì không giống như các trình ảo hóa, Docker không nhầm mục đích ảo hóa và chạy các Hệ điều hành và nhân khác nhau trên cùng một phần cứng. Mục đích chính của Docker là chứa các ứng dụng, vận chuyển và chạy chúng.

## Thùng chứa và máy ảo

MUMSHAD MANNAMBETH



Vì vậy, điều đó đưa chúng ta đến sự khác biệt giữa máy ảo và vùng chứa.

Điều gì đó mà chúng tôi có xu hướng làm, đặc biệt là những điều từ Ảo hóa.

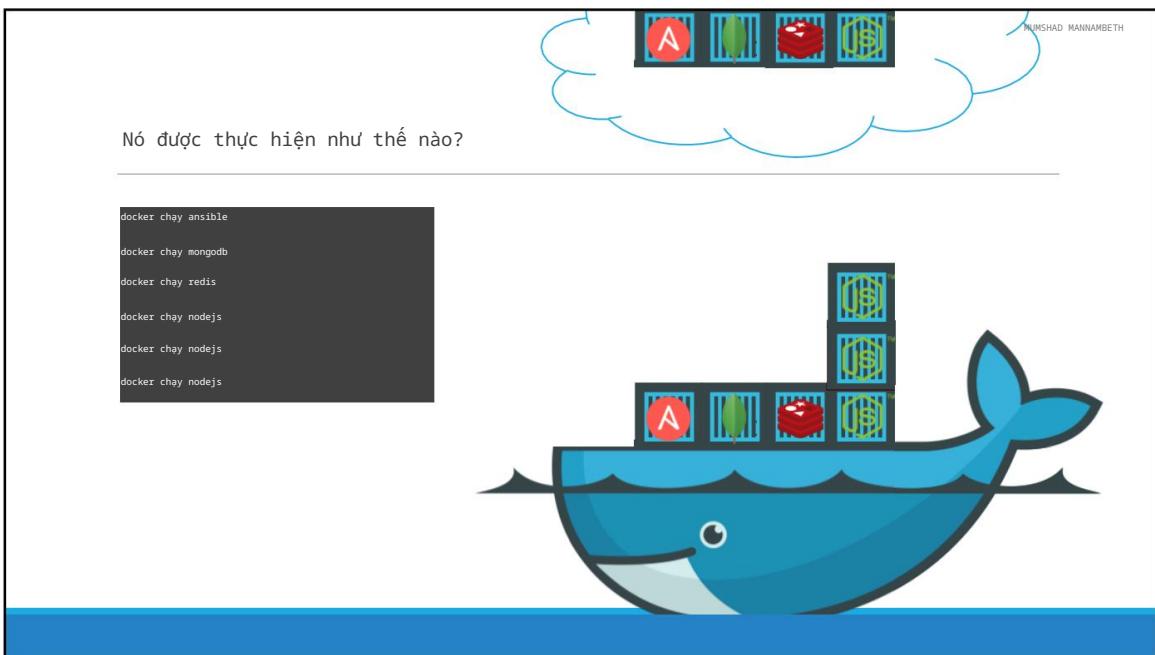
Như bạn có thể thấy ở bên phải, trong trường hợp Docker, chúng ta có cơ sở hạ tầng phần cứng cơ bản, sau đó là HĐH và Docker được cài đặt trên HĐH. Docker sau đó sẽ quản lý các vùng chứa chỉ chạy với các thư viện và phần phụ thuộc. Trong trường hợp Máy ảo, chúng tôi có HĐH trên phần cứng cơ bản, sau đó là Hypervisor như ESX hoặc loại ảo hóa nào đó và sau đó là máy ảo. Như bạn có thể thấy, mỗi máy ảo đều có hệ điều hành riêng bên trong, sau đó là các phần phụ thuộc và sau đó là ứng dụng.

Chi phí này khiến việc sử dụng tài nguyên cơ bản cao hơn vì có nhiều hệ điều hành ảo và kernel đang chạy. Các máy ảo cũng tiêu tốn dung lượng ổ đĩa cao hơn vì mỗi VM nặng và thường có kích thước Giga Byte, các thùng chứa docker có trọng lượng nhẹ và thường có kích thước Mega Byte.

Điều này cho phép các bộ chứa docker khởi động nhanh hơn, thường chỉ trong vài giây trong khi các máy ảo mà chúng tôi biết phải mất vài phút để khởi động vì nó cần khởi động toàn bộ hệ điều hành.

Điều quan trọng cần lưu ý là Docker có ít sự cô lập hơn vì có nhiều tài nguyên được chia sẻ giữa các vùng chứa như kernel, v.v. Trong khi đó, các máy ảo có sự cách ly hoàn toàn với nhau. Vì máy ảo không dựa vào hệ điều hành hoặc nhân cơ bản nên bạn có thể chạy các loại hệ điều hành khác nhau chẳng hạn như dựa trên linux hoặc windows dựa trên cùng một bộ ảo hóa.

Vì vậy, đây là một số khác biệt giữa hai.

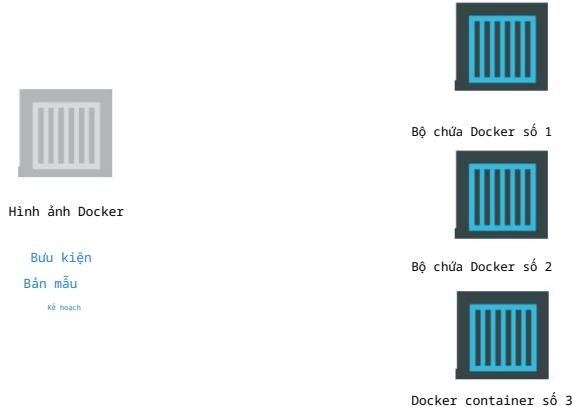


VẬY nó được thực hiện như thế nào? Hiện nay có rất nhiều phiên bản ứng dụng được đóng gói sẵn. Vì vậy, hầu hết các tổ chức đều có sản phẩm của họ được đóng gói và có sẵn trong cơ quan đăng ký docker công khai được gọi là dockerhub/hoặc docker store. Ví dụ: bạn có thể tìm thấy hình ảnh của hầu hết các hệ điều hành, cơ sở dữ liệu phổ biến cũng như các dịch vụ và công cụ khác. Khi bạn xác định được hình ảnh mình cần và cài đặt Docker trên máy chủ của mình..

Việc hiển thị một ngăn xếp ứng dụng cũng dễ dàng như chạy lệnh docker run với tên của hình ảnh. Trong trường hợp này, việc chạy lệnh docker run ansible sẽ chạy một phiên bản ansible trên máy chủ docker. Tương tự, chạy một phiên bản mongodb, redis và nodejs bằng lệnh docker run. Và sau đó khi bạn chạy nodejs, chỉ cần trỏ đến vị trí của kho lưu trữ mã trên máy chủ. Nếu chúng ta cần chạy nhiều phiên bản của dịch vụ web, bạn chỉ cần thêm bao nhiêu phiên bản tùy ý và định cấu hình một loại cân bằng tải nào đó ở phía trước. Trong trường hợp một trong các phiên bản bị lỗi, chỉ cần hủy phiên bản đó và khởi chạy một phiên bản mới. Có những giải pháp khác để xử lý những trường hợp như vậy mà chúng ta sẽ xem xét sau trong phần này.

khóa học.

## Vùng chứa so với hình ảnh



Chúng ta đã nói về hình ảnh và vùng chứa. Hãy hiểu sự khác biệt giữa hai người.

Hình ảnh là một gói hoặc mẫu, giống như mẫu VM mà bạn có thể đã làm việc với trong thế giới ảo hóa. Nó được sử dụng để tạo một hoặc nhiều container.

Các vùng chứa đang chạy các phiên bản của hình ảnh bị cô lập và có môi trường cũng như bộ quy trình riêng

Như chúng ta đã thấy trước đây, rất nhiều sản phẩm đã được cập bến. Trong trường hợp bạn không thể tìm thấy những gì mình đang tìm kiếm, bạn có thể tự tạo một hình ảnh và đẩy nó vào kho lưu trữ trung tâm Docker để nó có sẵn cho công chúng.

## Container Advantage



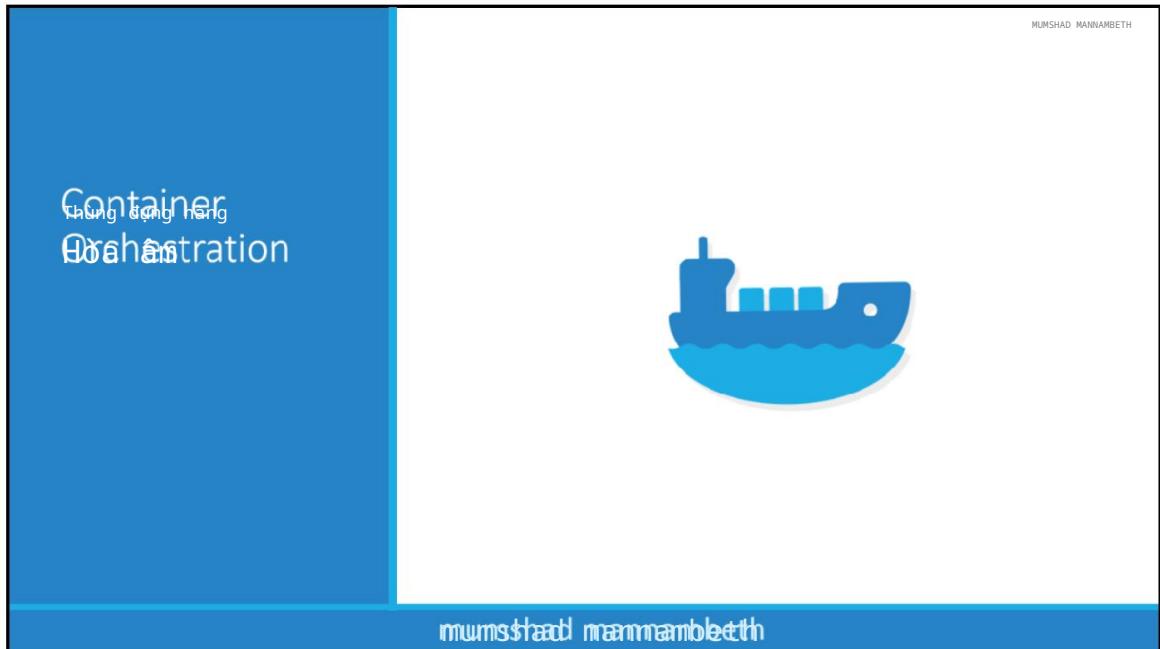
Nếu bạn nhìn vào nó, các nhà phát triển theo truyền thống đã phát triển các ứng dụng. Sau đó, họ giao nó cho nhóm Ops để triển khai và quản lý nó trong môi trường sản xuất. Họ thực hiện điều đó bằng cách cung cấp một bộ hướng dẫn như thông tin về cách thiết lập máy chủ, những điều kiện tiên quyết nào cần được cài đặt trên máy chủ và cách định cấu hình các phần phụ thuộc, v.v. Vì nhóm Ops không phát triển ứng dụng một mình họ phải vật lộn với việc thiết lập nó. Khi gặp sự cố, họ làm việc với các nhà phát triển để giải quyết.

Với Docker, phần lớn công việc liên quan đến việc thiết lập cơ sở hạ tầng hiện nay nằm trong tay các nhà phát triển dưới dạng tệp Docker. Hướng dẫn mà các nhà phát triển đã xây dựng trước đây để thiết lập cơ sở hạ tầng giờ đây có thể dễ dàng tập hợp lại thành Dockerfile để tạo hình ảnh cho ứng dụng của họ. Hình ảnh này hiện có thể chạy trên mọi nền tảng vùng chứa và được đảm bảo chạy theo cùng một cách ở mọi nơi. Vì vậy, nhóm Ops giờ đây có thể chỉ cần sử dụng hình ảnh để triển khai ứng dụng. Vì hình ảnh đã hoạt động khi nhà phát triển xây dựng nó và các hoạt động không sửa đổi nó nên nó tiếp tục hoạt động như cũ khi được triển khai trong sản xuất.

## Tìm hiểu thêm về container

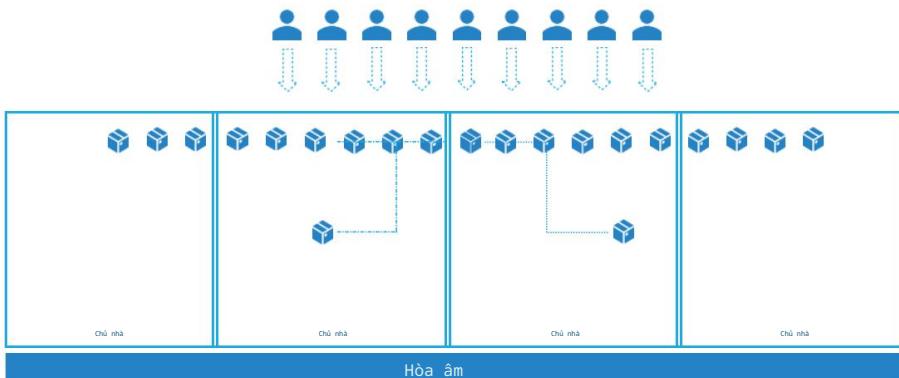


Để tìm hiểu thêm về vùng chứa, hãy xem các khóa học khác của tôi - Docker dành cho người mới bắt đầu tuyệt đối và Docker Swarm là nơi bạn có thể tìm hiểu và thực hành các lệnh docker cũng như tạo tệp docker. Bài giảng này về Container và Docker đến đây là kết thúc. Hẹn gặp lại các bạn trong bài giảng tiếp theo.



Trong bài giảng này, chúng ta sẽ nói về Điều phối vùng chứa.

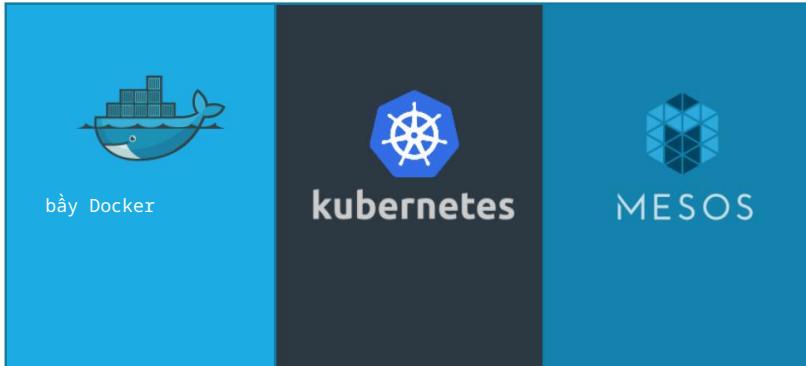
## Điều phối vùng chứa



Vậy là chúng ta đã tìm hiểu về vùng chứa và giờ chúng ta đã đóng gói ứng dụng của mình vào vùng chứa docker. Nhưng tiếp theo là gì? Làm thế nào để bạn chạy nó trong sản xuất? Điều gì sẽ xảy ra nếu ứng dụng của bạn dựa trên các vùng chứa khác như cơ sở dữ liệu hoặc dịch vụ nhắn tin hoặc các dịch vụ phụ trợ khác? Điều gì sẽ xảy ra nếu số lượng người dùng tăng lên và bạn cần mở rộng quy mô ứng dụng của mình? Bạn cũng muốn giảm quy mô khi tải giảm.

Để kích hoạt các chức năng này, bạn cần có một nền tảng cơ bản với một bộ tài nguyên. Nền tảng cần điều phối kết nối giữa các vùng chứa và tự động tăng hoặc giảm quy mô dựa trên tải. Toàn bộ quá trình tự động triển khai và quản lý vùng chứa này được gọi là Điều phối vùng chứa.

## Công nghệ điều phối



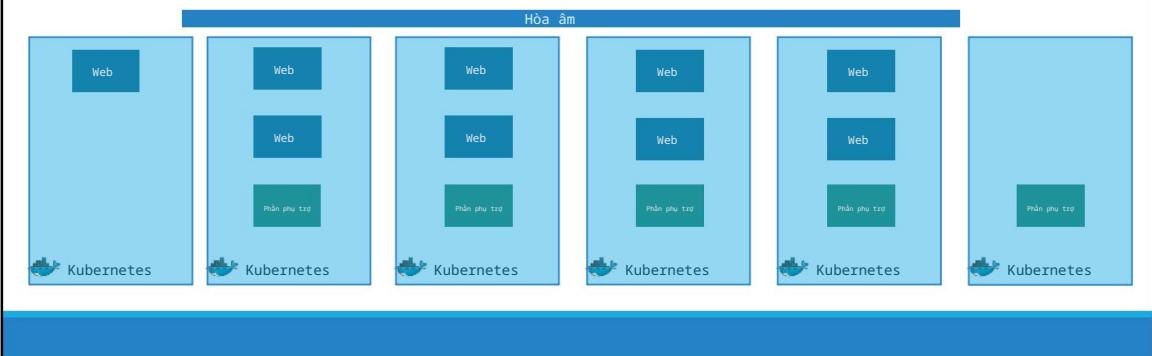
Kubernetics do đó là một công nghệ điều phối vùng chứa. Hiện nay có rất nhiều công nghệ như vậy

- Docker có công cụ riêng gọi là Docker Swarm.

Kubernetes từ Google và Mesos từ Apache. Mặc dù Docker Swarm thực sự dễ cài đặt và bắt đầu nhưng nó thiếu một số tính năng tự động mở rộng quy mô nâng cao cần thiết cho các ứng dụng phức tạp. Mặt khác, Mesos khá khó cài đặt và bắt đầu nhưng lại hỗ trợ nhiều tính năng nâng cao. Kubernetes - được cho là phổ biến nhất - hơi khó thiết lập và bắt đầu nhưng cung cấp nhiều tùy chọn để tùy chỉnh việc triển khai và hỗ trợ triển khai các kiến trúc phức tạp.

Kubernetics hiện được hỗ trợ trên tất cả các nhà cung cấp dịch vụ đám mây công cộng như GCP, Azure và AWS và dự án kubernetes là một trong những dự án được xếp hạng hàng đầu trên Github.

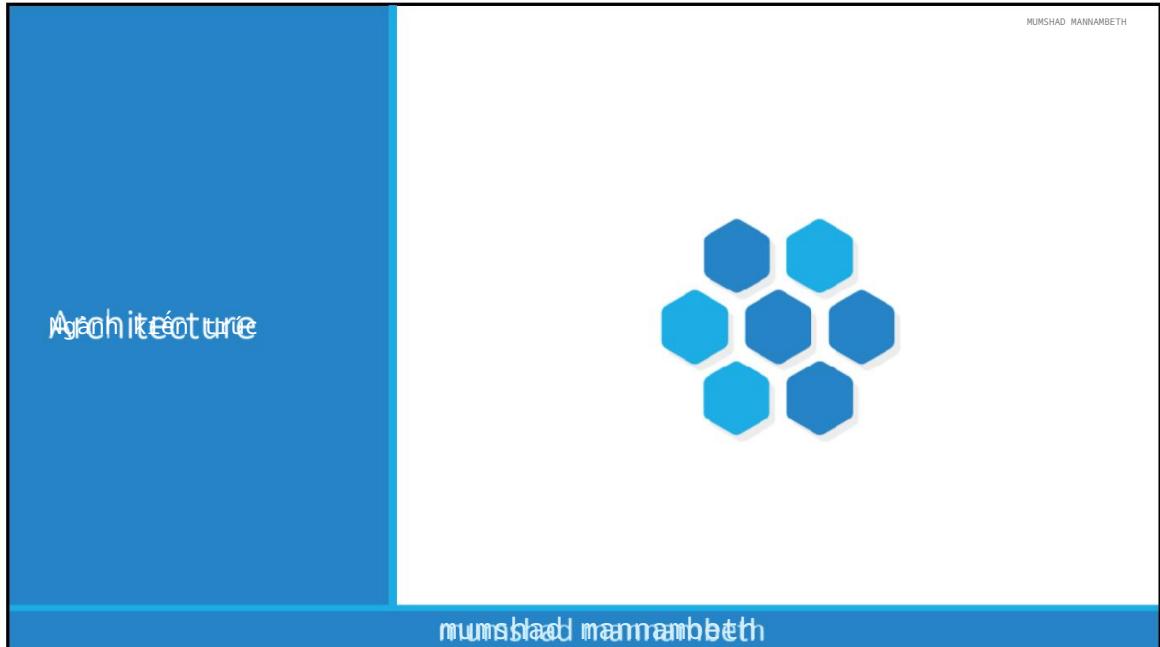
## Lợi thế của Kubernetes



Có nhiều lợi ích khác nhau của việc điều phối container. Ứng dụng của bạn hiện có tính khả dụng cao vì lỗi phần cứng không làm ứng dụng của bạn ngừng hoạt động vì bạn có nhiều phiên bản ứng dụng đang chạy trên các nút khác nhau. Lưu lượng người dùng được cân bằng tải trên các vùng chứa khác nhau. Khi nhu cầu tăng lên, hãy triển khai nhiều phiên bản ứng dụng hơn một cách liền mạch và chỉ trong vài giây và chúng tôi có khả năng thực hiện điều đó ở cấp độ dịch vụ. Khi chúng tôi hết tài nguyên phần cứng, hãy tăng/giảm quy mô số lượng nút mà không cần phải gỡ bỏ ứng dụng. Và thực hiện tất cả những điều này một cách dễ dàng với một tập hợp các tệp cấu hình đối tượng khai báo.

Và đó là **kubernetes..**

Và ĐÓ LÀ Kubernetes. Đây là công nghệ Điều phối vùng chứa được sử dụng để điều phối việc triển khai và quản lý 100 và 1000 vùng chứa trong môi trường phân cụm. Đừng lo lắng nếu bạn không hiểu hết những gì vừa nói, trong các bài giảng sắp tới, chúng ta sẽ xem xét sâu hơn về kiến trúc và các khái niệm khác nhau xung quanh Kubernetes. Đó là tất cả cho bài giảng này, cảm ơn bạn đã lắng nghe và tôi sẽ gặp lại bạn trong bài giảng tiếp theo.



Trước khi bắt đầu thiết lập cụm kubernetes, điều quan trọng là phải hiểu một số khái niệm cơ bản. Điều này nhằm hiểu các thuật ngữ mà chúng ta sẽ gặp khi thiết lập cụm kubernetes.

## Nút (Minion)

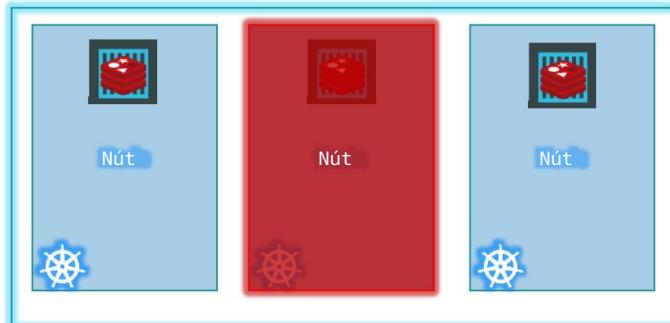


Hãy bắt đầu với Nodes. Nút là một máy - vật lý hoặc ảo - được cài đặt kubernetes trên đó. Nút là một máy công nhân và đây là nơi các thùng chứa sẽ được đưa ra bởi kubernetes.

Trước đây nó còn được gọi là Minions. Vì vậy, ở đây bạn có thể sử dụng những thuật ngữ này có thể thay đổi được.

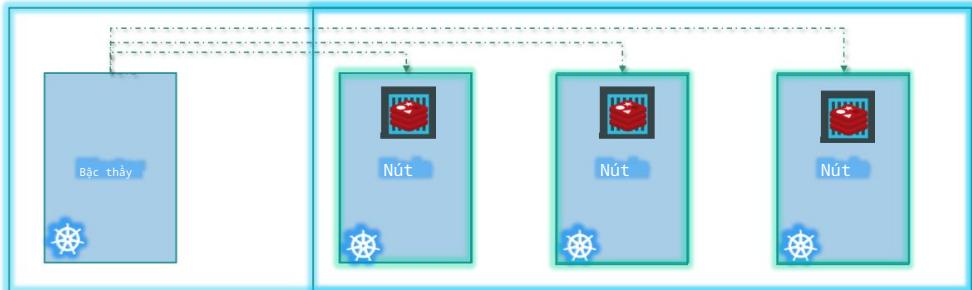
Nhưng điều gì sẽ xảy ra nếu nút mà ứng dụng của chúng ta đang chạy bị lỗi? Chà, rõ ràng là ứng dụng của chúng tôi bị hỏng. Vì vậy, bạn cần phải có nhiều hơn một nút.

## Cụm



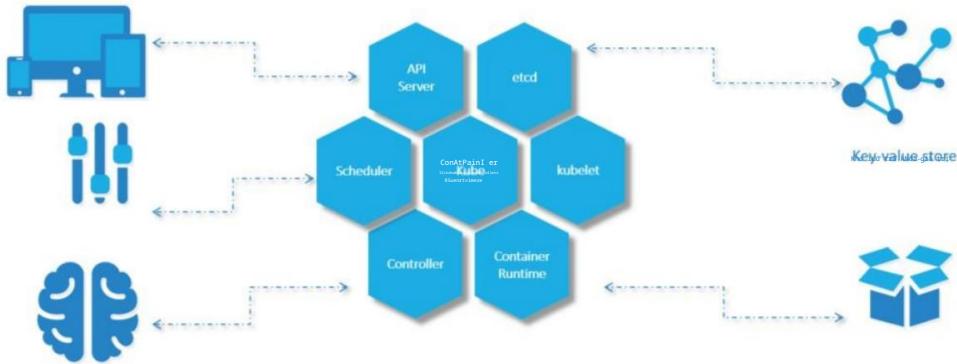
Một cụm là một tập hợp các nút được nhóm lại với nhau. Bằng cách này, ngay cả khi một nút bị lỗi, ứng dụng của bạn vẫn có thể truy cập được từ các nút khác. Hơn nữa, việc có nhiều nút cũng giúp chia sẻ tải.

## Bậc thầy



Bây giờ chúng ta có một cụm, nhưng ai chịu trách nhiệm quản lý cụm đó? Thông tin về các thành viên của cụm có được lưu trữ không? Các nút được giám sát như thế nào? Khi một nút bị lỗi, bạn làm cách nào để di chuyển khối lượng công việc của nút bị lỗi sang nút công nhân khác? Đó là lúc Master xuất hiện. Master là một nút khác được cài đặt Kubernetes trong đó và được định cấu hình là Master. Master theo dõi các nút trong cụm và chịu trách nhiệm điều phối thực tế các vùng chứa trên các nút công nhân.

## Các thành phần



Khi cài đặt Kubernetes trên Hệ thống, thực tế là bạn đang cài đặt các thành phần sau. Một máy chủ API. Một dịch vụ ETCD. Một dịch vụ kubelet. Thời gian chạy, bộ điều khiển và bộ lập lịch của vùng chứa.

Máy chủ API hoạt động như giao diện người dùng cho kubernetes. Người dùng, thiết bị quản lý, giao diện dòng lệnh đều nói chuyện với máy chủ API để tương tác với cụm kubernetes.

Tiếp theo là kho khóa ETCD. ETCD là kho lưu trữ khóa-giá trị đáng tin cậy được phân phối được kubernetes sử dụng để lưu trữ tất cả dữ liệu dùng để quản lý cụm. Hãy nghĩ theo cách này, khi bạn có nhiều nút và nhiều nút chính trong cụm của mình, etcd sẽ lưu trữ tất cả thông tin đó trên tất cả các nút trong cụm theo cách phân tán. ETCD chịu trách nhiệm triển khai các khóa trong cụm để đảm bảo không có xung đột giữa các Master.

Bộ lập lịch chịu trách nhiệm phân phối công việc hoặc vùng chứa trên nhiều nút. Nó tìm kiếm các vùng chứa mới được tạo và gán chúng cho Nút.

Người điều khiển là bộ não đằng sau việc điều phối. Họ chịu trách nhiệm thông báo và phản hồi khi các nút, vùng chứa hoặc điểm cuối gặp sự cố. Người kiểm soát đưa ra quyết định đưa ra các vùng chứa mới trong những trường hợp như vậy.

Thời gian chạy vùng chứa là phần mềm cơ bản được sử dụng để chạy vùng chứa. Trong trường hợp của chúng tôi, đó là Docker.

Và cuối cùng kubelet là tác nhân chạy trên mỗi nút trong cụm. Tác nhân chịu trách nhiệm đảm bảo rằng các thùng chứa đang chạy trên các nút như mong đợi.

## Nút chính và nút công nhân



Cho đến nay chúng ta đã thấy hai loại máy chủ - Master và Worker và một tập hợp các thành phần tạo nên Kubernetes. Nhưng làm thế nào các thành phần này được phân phối trên các loại máy chủ khác nhau. Nói cách khác, làm thế nào để một máy chủ trở thành chủ và máy chủ kia trở thành nô lệ?

Nút công nhân (hoặc tay sai), như nó còn được gọi, là nơi chứa các vùng chứa.

Ví dụ: bộ chứa Docker và để chạy bộ chứa docker trên hệ thống, chúng ta cần cài đặt thời gian chạy bộ chứa. Và đó là thời gian chạy của container bị giảm. Trong trường hợp này nó xảy ra là Docker. Cái này KHÔNG PHẢI là docker, có sẵn các lựa chọn thay thế thời gian chạy container khác như Rocket hoặc CRI0. Nhưng trong suốt khóa học này, chúng ta sẽ sử dụng Docker làm thời gian chạy vùng chứa.

Máy chủ chính có kube-apiserver và đó là điều khiến nó trở thành máy chủ chính.

Tương tự, các nút công nhân có tác nhân kubelet chịu trách nhiệm tương tác với nút chủ để cung cấp thông tin tình trạng của nút thợ và thực hiện các hành động do chủ yêu cầu trên nút thợ.

Tất cả thông tin thu thập được sẽ được lưu trữ trong kho lưu trữ khóa-giá trị trên Master. Kho lưu trữ giá trị chính dựa trên khung etcd phổ biến như chúng ta vừa thảo luận.

Bản gốc cũng có trình quản lý bộ điều khiển và bộ lập lịch.

Ngoài ra còn có các thành phần khác, nhưng bây giờ chúng ta sẽ dùng ở đó. Lý do chúng tôi thực hiện điều này là để hiểu những thành phần nào cấu thành nút chính và nút thợ. Điều này sẽ giúp chúng tôi cài đặt và định cấu hình các thành phần phù hợp trên các hệ thống khác nhau khi chúng tôi thiết lập cơ sở hạ tầng của mình.

## kubectl

```
kubectl chạy hello-minikube
```



```
thông tin cụm kubectl
```



```
kubectl lấy nút
```

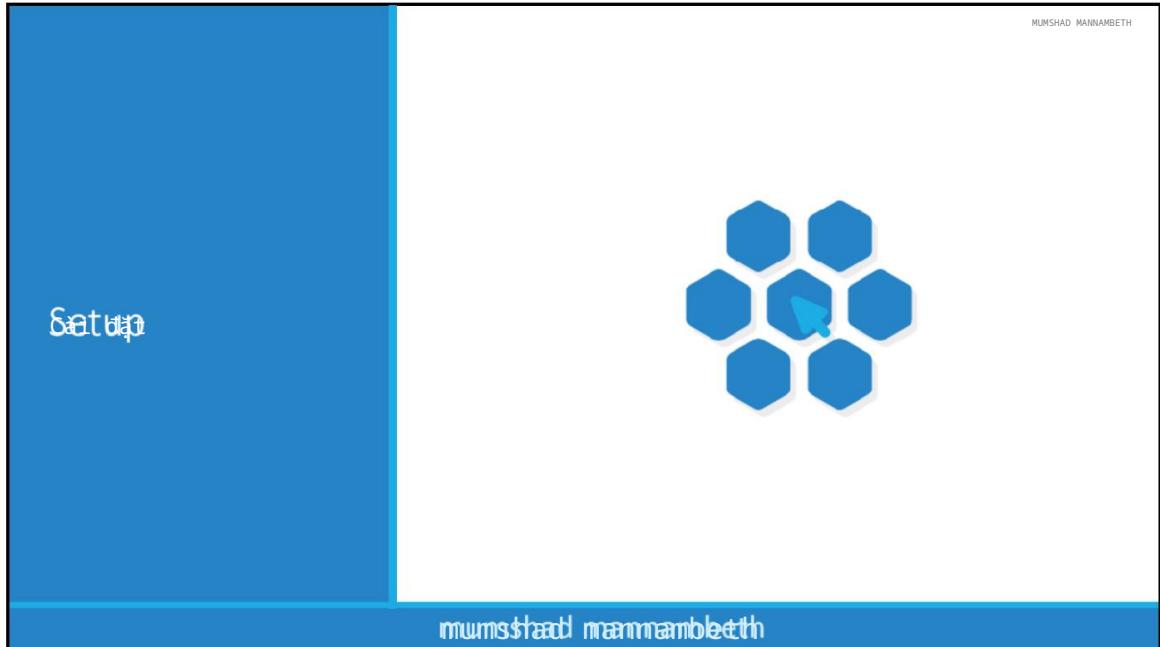


Và cuối cùng, chúng ta cũng cần tìm hiểu một chút về MỘT trong số các tiện ích dòng lệnh được gọi là công cụ dòng lệnh kube hoặc kubectl hoặc kube control như nó còn được gọi.

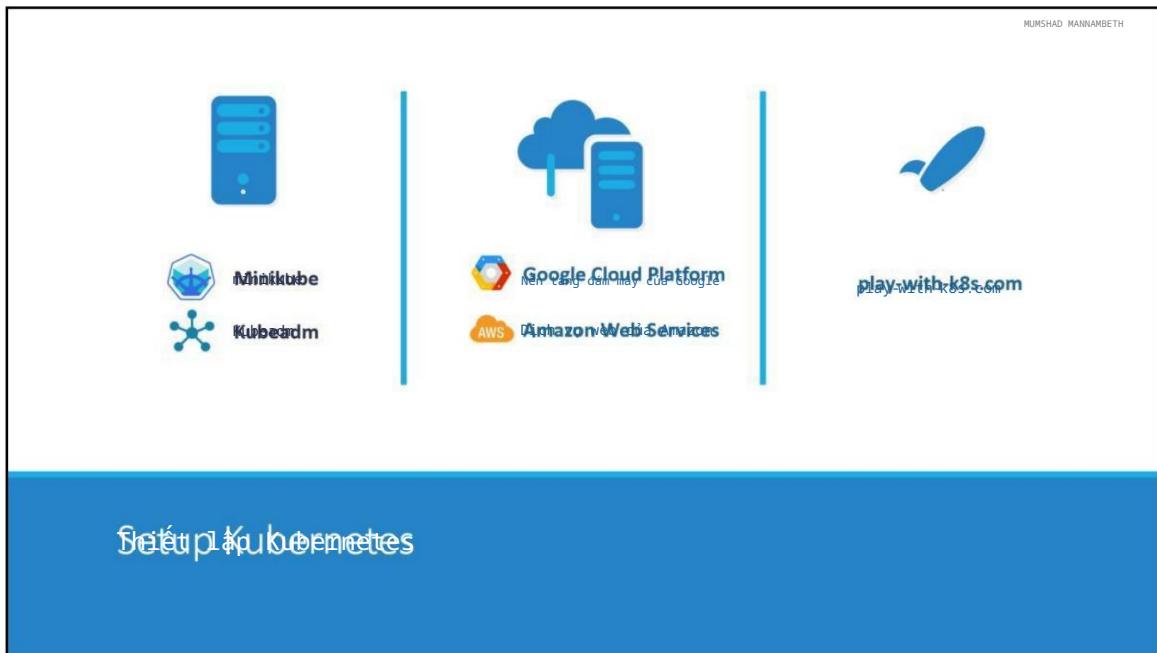
Công cụ điều khiển kube được sử dụng để triển khai và quản lý các ứng dụng trên cụm kubernetes, để lấy thông tin cụm, lấy trạng thái của các nút trong cụm và nhiều thứ khác.

Lệnh kubectl run được sử dụng để triển khai một ứng dụng trên cluster. Lệnh kubectl cluster-info được sử dụng để xem thông tin về cụm và lệnh kubectl get pod được sử dụng để liệt kê tất cả các phần nút của cụm. Đó là tất cả những gì chúng ta cần biết bây giờ và chúng ta sẽ tiếp tục tìm hiểu thêm các lệnh trong suốt khóa học này. Chúng ta sẽ khám phá thêm các lệnh với kubectl khi tìm hiểu các khái niệm liên quan.

Bây giờ, chỉ cần nhớ các lệnh chạy, thông tin cụm và nhận nút và điều đó sẽ giúp chúng ta vượt qua một số phòng thí nghiệm đầu tiên.



Trong bài giảng này, chúng ta sẽ xem xét các tùy chọn khác nhau có sẵn trong việc xây dựng cụm Kubernetes từ đầu.



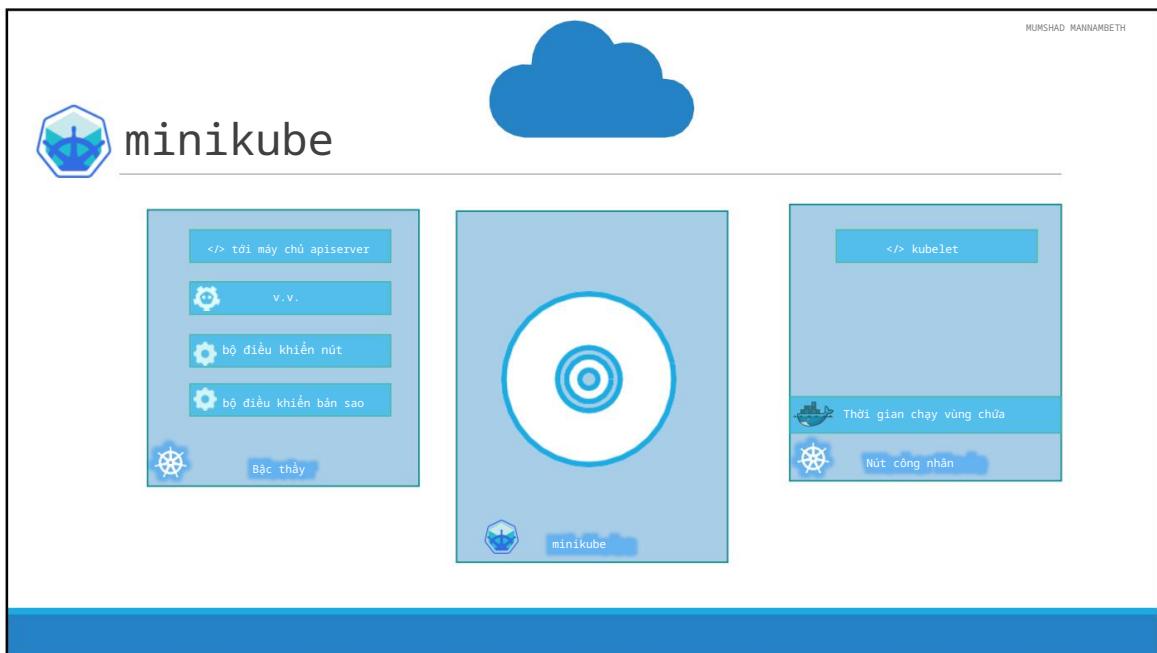
Có rất nhiều cách để thiết lập Kubernetes. Chúng tôi có thể tự thiết lập cục bộ trên máy tính xách tay hoặc máy ảo của mình bằng các giải pháp như Minikube và Kubeadmin.

Minikube là công cụ dùng để thiết lập một phiên bản Kubernetes duy nhất trong thiết lập Tất cả trong một và kubeadmin là công cụ dùng để định cấu hình kubernetes trong thiết lập nhiều nút. Chúng ta sẽ xem xét thêm về điều đó một chút.

Ngoài ra còn có các giải pháp lưu trữ sẵn có để thiết lập kubernetes trên đám mây môi trường như GCP và AWS. Chúng tôi cũng sẽ có một số bản demo xung quanh những điều đó.

Và cuối cùng, nếu bạn không có tài nguyên hoặc nếu bạn không muốn gặp rắc rối khi tự mình thiết lập tất cả và bạn chỉ muốn có được một cụm kubernetes ngay lập tức để chơi, hãy kiểm tra [play-with-k8s.com](http://play-with-k8s.com). Tôi cũng có một bản demo về điều này.

Vì vậy, hãy thoải mái lựa chọn một trong đó phù hợp với bạn. Bạn không cần phải xem qua tất cả các bản demo, hãy chọn những bản demo phù hợp nhất với nhu cầu của bạn dựa trên thời gian và nguồn lực của bạn.

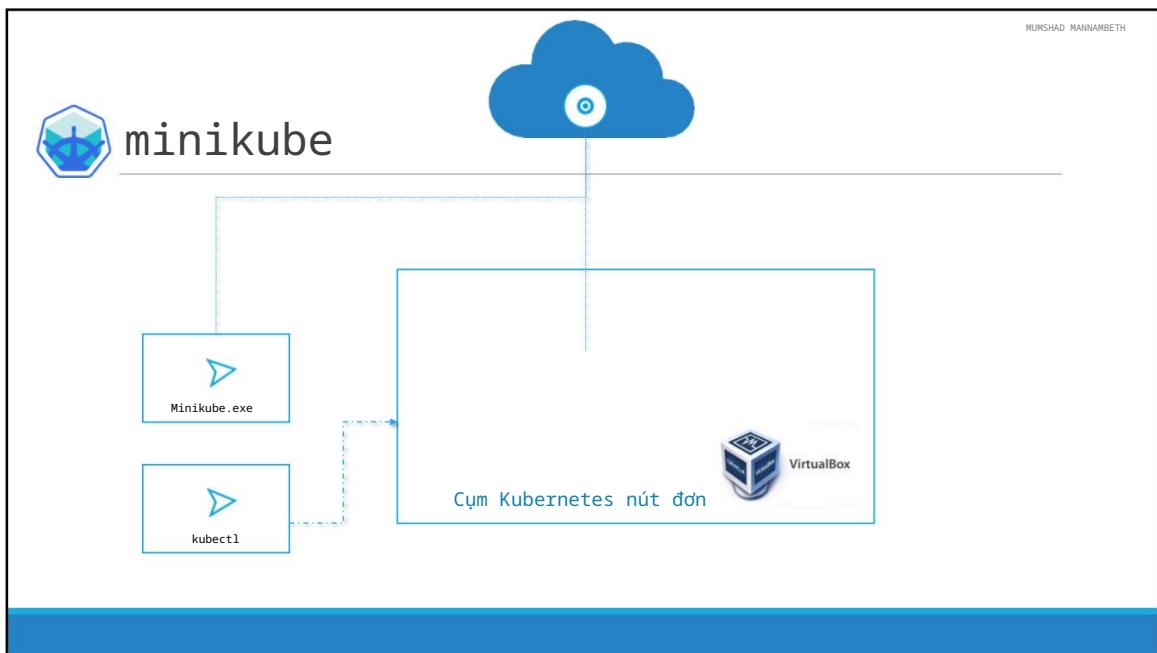


Chúng ta sẽ bắt đầu với Minikube, đây là cách dễ nhất để bắt đầu với Kubernetes trên hệ thống cục bộ. Nếu bạn không quan tâm đến Minikube thì bây giờ là thời điểm tốt để bỏ qua bài giảng này. Trước khi đi vào phần demo, bạn nên hiểu cách thức hoạt động của nó.

Trước đó chúng ta đã nói về các thành phần khác nhau của Kubernetes tạo nên nút Master và nút công nhân, chẳng hạn như apiserver, kho lưu trữ giá trị khóa etcd, bộ điều khiển và bộ lập lịch trên nút chính và kubelets cũng như thời gian chạy vùng chứa trên nút công nhân. Sẽ mất rất nhiều thời gian và công sức để tự mình thiết lập và cài đặt tất cả các thành phần khác nhau này trên các hệ thống khác nhau.

Minikube gói tất cả các thành phần khác nhau này vào một hình ảnh duy nhất cung cấp cho chúng tôi cụm kubernetes nút đơn được định cấu hình sẵn để chúng tôi có thể bắt đầu sau vài phút.

Toàn bộ gói được đóng gói thành một ảnh ISO và có sẵn trực tuyến để tải xuống.



Bây giờ bạn KHÔNG PHẢI tự tải xuống. Minikube cung cấp tiện ích dòng lệnh thực thi sẽ TỰ ĐỘNG tải xuống ISO và triển khai nó trong nền tảng ảo hóa như Oracle Virtualbox hoặc Vmware fusion. Vì vậy, bạn phải cài đặt Hypervisor trên hệ thống của mình. Đối với windows, bạn có thể sử dụng Virtualbox hoặc Hyper-V và đối với Linux, hãy sử dụng Virtualbox hoặc KVM.

Và cuối cùng để tương tác với cụm kubernetes, bạn phải cài đặt công cụ dòng lệnh kubectl kubernetes trên máy của mình. Vì vậy, bạn cần 3 thứ để làm việc này, bạn phải cài đặt bộ ảo hóa, cài đặt kubectl và cài đặt tệp thực thi minikube trên hệ thống của bạn.

Vì vậy, hãy đi vào bản demo

MUMSHAD MANNAMBETH

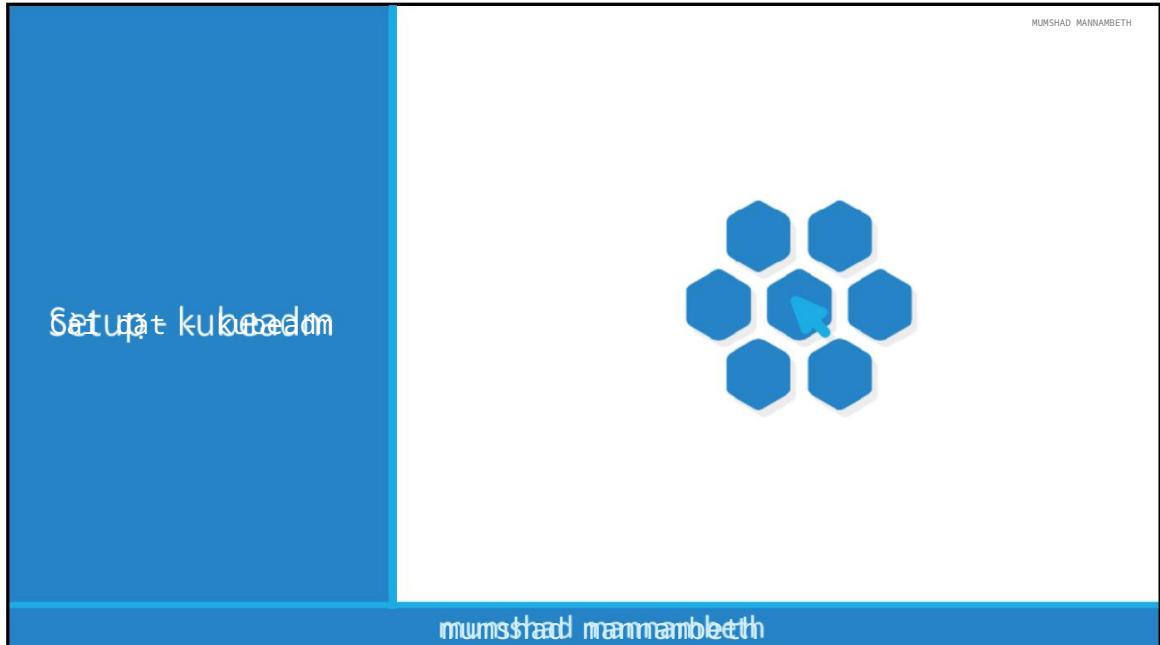


Thử nghiệm

---

minikube

Bài giảng này thế là xong, chúng ta hãy xem bản demo và xem nó hoạt động như thế nào.



Xin chào và chào mừng bạn đến với bài giảng này về cách thiết lập Kubernetes với kubeadm.  
Trong bài giảng này, chúng ta sẽ xem xét công cụ kubeadm có thể được sử dụng để  
khởi động cụm kubernetes.

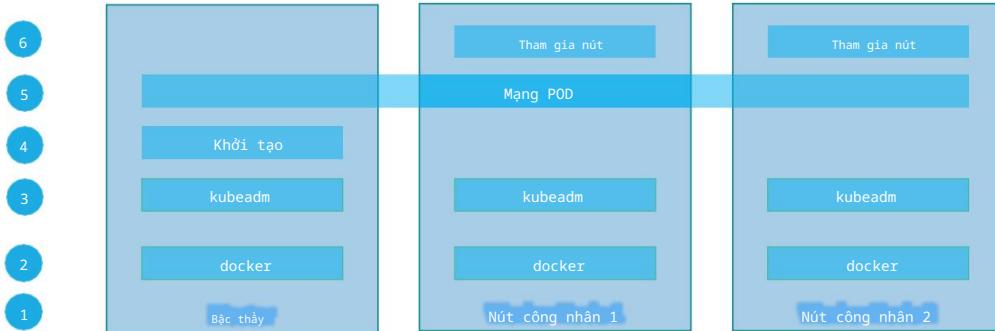
# kubeadm



Với tiện ích minikube, bạn chỉ có thể thiết lập một cụm kubernetes một nút. Công cụ kubeadmin giúp chúng ta thiết lập một cụm nhiều nút với máy chính và máy công nhân trên các máy riêng biệt. Việc cài đặt tất cả các thành phần khác nhau này một cách riêng lẻ trên các nút khác nhau và sửa đổi các tệp cấu hình để làm cho nó hoạt động là một công việc tẻ nhạt.

Công cụ Kubeadmin giúp chúng ta thực hiện tất cả những điều đó một cách rất dễ dàng.

# bước



Hãy thực hiện các bước - Trước tiên, bạn phải tạo nhiều hệ thống hoặc máy ảo để định cấu hình một cụm. Chúng tôi sẽ xem cách thiết lập máy tính xách tay của bạn để làm việc đó nếu bạn chưa quen với nó. Sau khi hệ thống được tạo, hãy chỉ định một hệ thống làm nút chính và các hệ thống khác làm nút công nhân.

Bước tiếp theo là cài đặt thời gian chạy vùng chứa trên máy chủ. Chúng tôi sẽ sử dụng Docker, vì vậy chúng tôi phải cài đặt Docker trên tất cả các nút.

Bước tiếp theo là cài đặt công cụ kubeadmin trên tất cả các nút. Công cụ kubeadmin giúp chúng ta khởi động giải pháp Kubernetes bằng cách cài đặt và định cấu hình tất cả các thành phần cần thiết ở các nút phù hợp.

Bước thứ ba là khởi tạo máy chủ Master. Trong quá trình này, tất cả các thành phần cần thiết sẽ được cài đặt và cấu hình trên máy chủ chính. Bằng cách đó, chúng ta có thể bắt đầu cấu hình cấp cụm từ máy chủ chính.

Sau khi khởi tạo nút chính và trước khi nối các nút công nhân với nút chính, chúng tôi phải đảm bảo rằng các điều kiện tiên quyết của mạng được đáp ứng. Kết nối mạng bình thường giữa các hệ thống là không đủ cho việc này. Kubernetes yêu cầu một chương trình đặc biệt

mạng giữa nút chính và nút công nhân được gọi là mạng POD. Chúng tôi sẽ tìm hiểu thêm về mạng này trong phần mạng ở phần sau của khóa học này. Hiện tại, chúng tôi sẽ chỉ làm theo các hướng dẫn có sẵn để cài đặt và thiết lập tính năng này trong môi trường của chúng tôi.

Bước cuối cùng là nối các nút công nhân với nút chính. Sau đó, chúng tôi đã sẵn sàng khởi chạy ứng dụng của mình trong môi trường kubernetes.

MUMSHAD MANNAMBETH



Thử nghiệm

---

kubeadm

Bây giờ chúng ta sẽ thấy bản demo cách thiết lập kubernetes bằng công cụ kubeadm trong môi trường cục bộ của chúng ta.

MUMSHAD MANAMBETH



## Thử nghiệm

Nền tảng đám mây của Google

Bây giờ chúng ta sẽ thấy bản demo cách thiết lập kubernetes bằng công cụ kubeadmin trong môi trường cục bộ của chúng ta.

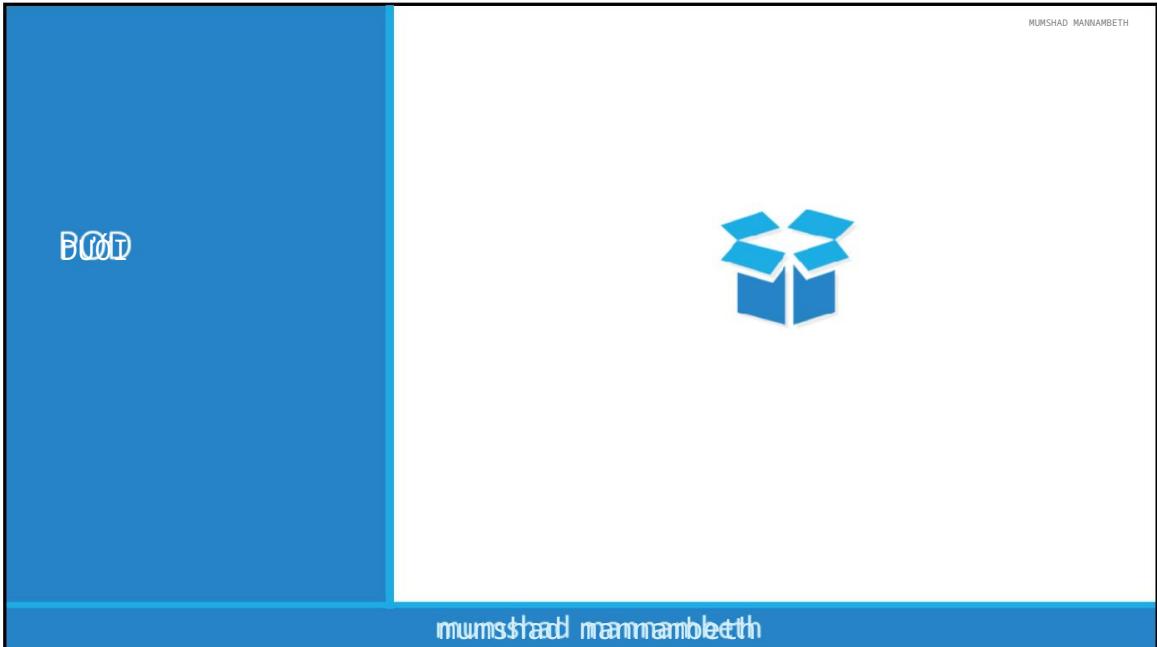
MUMSHAD MANAMBETH



Thử nghiệm

[play-with-k8s.com](http://play-with-k8s.com)

Bây giờ chúng ta sẽ thấy bản demo cách thiết lập kubernetes bằng công cụ kubeadmin trong môi trường cục bộ của chúng ta.



Xin chào và chào mừng bạn đến với bài giảng này về Kubernetes POD. Trong bài giảng này, chúng ta sẽ thảo luận về Kubernetes POD.

## ເນື້ອລັບເຫັນທີ່



Docker Image

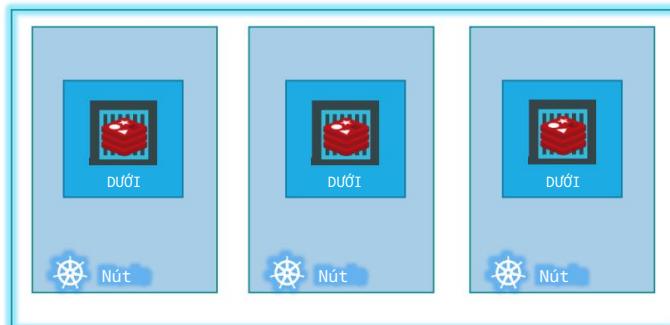


Kubernetes Cluster



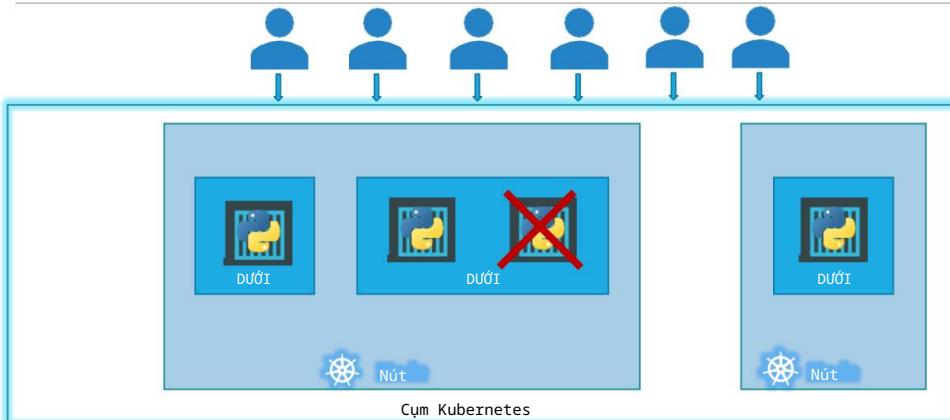
Trước khi tìm hiểu POD, chúng tôi muốn giả định rằng những điều sau đây đã được thiết lập. Tại thời điểm này, chúng tôi giả định rằng ứng dụng đã được phát triển và tích hợp vào Docker Images và nó có sẵn trên kho Docker như Docker hub, vì vậy Kubernetes có thể kéo nó xuống. Chúng tôi cũng giả định rằng cụm Kubernetes đã được thiết lập và đang hoạt động. Đây có thể là thiết lập một nút hoặc thiết lập nhiều nút, không thành vấn đề. Tất cả các dịch vụ cần phải ở trạng thái đang chạy.

## DƯỚI



Như chúng ta đã thảo luận trước đây, với kubernetes, mục đích cuối cùng của chúng ta là triển khai ứng dụng của mình dưới dạng vùng chứa trên một tập hợp các máy được định cấu hình làm nút công nhân trong một cụm. Tuy nhiên, kubernetes không triển khai các container trực tiếp trên các nút công nhân. Các thùng chứa được gói gọn trong một đối tượng Kubernetes được gọi là POD. POD là một phiên bản duy nhất của một ứng dụng. POD là đối tượng nhỏ nhất mà bạn có thể tạo trong kubernetes.

## DƯỚI

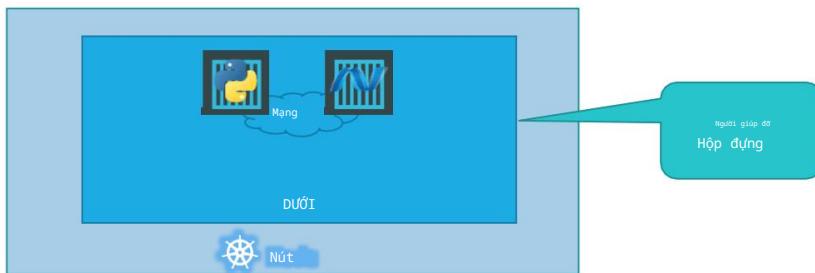


Ở đây, chúng ta thấy trường hợp đơn giản nhất trong số các trường hợp đơn giản nhất là bạn có một cụm kubernetes nút duy nhất với một phiên bản ứng dụng duy nhất của bạn đang chạy trong một bộ chứa docker duy nhất được gói gọn trong POD. Điều gì sẽ xảy ra nếu số lượng người dùng truy cập ứng dụng của bạn tăng lên và bạn cần mở rộng quy mô ứng dụng của mình? Bạn cần thêm các phiên bản bổ sung của ứng dụng web của mình để chia sẻ tải. Bây giờ, bạn có muốn thêm phiên bản bổ sung không? Chúng tôi có đưa ra một phiên bản vùng chứa mới trong cùng một POD không? KHÔNG! Chúng tôi tạo một POD mới hoàn toàn với một phiên bản mới của cùng một ứng dụng. Như bạn có thể thấy, hiện tại chúng tôi có hai phiên bản ứng dụng web chạy trên hai POD riêng biệt trên cùng một hệ thống hoặc nút kubernetes.

Điều gì sẽ xảy ra nếu cơ sở người dùng THÊM tăng lên và nút hiện tại của bạn không có đủ dung lượng? Vâng, THÌ bạn luôn có thể triển khai các POD bổ sung trên một nút mới trong cụm. Bạn sẽ có một nút mới được thêm vào cụm để mở rộng dung lượng vật lý của cụm. VẬY, điều tôi đang cố gắng minh họa trong trang trình bày này là POD thường có mối quan hệ một-một với các vùng chứa chạy ứng dụng của bạn. Để mở rộng quy mô UP, bạn tạo POD mới và để thu nhỏ quy mô, bạn xóa POD. Bạn không thêm các vùng chứa bổ sung vào POD hiện có để mở rộng quy mô ứng dụng của mình. Ngoài ra, nếu bạn đang thắc mắc về cách chúng tôi triển khai tất cả những điều này và cách chúng tôi đạt được cân bằng tải giữa các vùng chứa, v.v., chúng tôi sẽ đề cập đến tất cả những điều đó trong bài giảng sau. Hiện tại chúng tôi CHỈ đang cố gắng

hiểu các khái niệm cơ bản.

## POD nhiều container



Bây giờ chúng ta vừa nói rằng POD thường có mối quan hệ một-một với các vùng chứa, tuy nhiên, chúng ta có bị hạn chế chỉ có một vùng chứa duy nhất trong một POD không? KHÔNG! Một POD CÓ THỂ có nhiều vùng chứa, ngoại trừ thực tế là chúng thường không phải là nhiều vùng chứa cùng loại. Như chúng ta đã thảo luận trong trang trình bày trước, nếu mục đích của chúng ta là mở rộng quy mô ứng dụng thì chúng ta sẽ cần tạo thêm các POD. Nhưng đôi khi bạn có thể gặp một tình huống là bạn có một vùng chứa trợ giúp, có thể đang thực hiện một số loại tác vụ hỗ trợ cho ứng dụng web của chúng tôi, chẳng hạn như xử lý dữ liệu do người dùng nhập, xử lý tệp do người dùng tải lên, v.v. và bạn muốn các vùng chứa trợ giúp này để sống cùng với vùng chứa ứng dụng của bạn. Trong trường hợp đó, bạn CÓ THỂ có cả hai vùng chứa này thuộc cùng một POD, để khi một vùng chứa ứng dụng mới được tạo, trình trợ giúp cũng được tạo và khi nó chết thì trình trợ giúp cũng chết vì chúng là một phần của cùng một POD. Hai vùng chứa cũng có thể giao tiếp trực tiếp với nhau bằng cách gọi nhau là 'localhost' vì chúng có chung không gian tên mạng.Thêm vào đó, họ cũng có thể dễ dàng chia sẻ cùng một không gian lưu trữ.

MUMSHAD MANNAMBETH

## POD một lần nữa!

docker chạy ứng dụng python  
trình trợ giúp chạy docker -link app1  
trình trợ giúp chạy docker -link app2  
trình trợ giúp chạy docker -link app3  
trình trợ giúp chạy docker -link app4

| Ứng dụng          | Khởi lượng | xự giúp |
|-------------------|------------|---------|
| Ứng dụng Python11 | tập 1      |         |
| Ứng dụng Python22 | Tập 2      |         |

Lưu ý: Tôi tránh các chi tiết về mạng và cân bằng tải để giải thích đơn giản.

Nếu bạn vẫn còn nghi ngờ về chủ đề này (tôi sẽ hiểu nếu bạn nghi ngờ vì tôi đã làm như vậy trong lần đầu tiên tôi học những khái niệm này), chúng ta có thể thực hiện một bước khác để hiểu POD từ một góc độ khác. Trong giây lát, hãy loại bỏ Kubernetes ra khỏi hệ thống của chúng ta thảo luận và nói về các container Docker đơn giản. Giả sử chúng ta đang phát triển một quy trình hoặc một tập lệnh để triển khai ứng dụng của mình trên máy chủ Docker. Sau đó, trước tiên chúng tôi chỉ cần triển khai ứng dụng của mình bằng cách sử dụng lệnh `docker run python-app` đơn giản và ứng dụng sẽ chạy tốt và người dùng của chúng tôi có thể truy cập nó. Khi tải tảng lên, chúng tôi sẽ triển khai nhiều phiên bản ứng dụng hơn bằng cách chạy lệnh `docker run` nhiều lần nữa. Điều này hoạt động tốt và tất cả chúng tôi đều hạnh phúc. Bây giờ, trong tương lai, ứng dụng của chúng tôi sẽ được phát triển hơn nữa, trải qua những thay đổi về kiến trúc, phát triển và trở nên phức tạp. Hiện tại, chúng tôi có các vùng chứa trợ giúp mới giúp các ứng dụng web của chúng tôi xử lý hoặc tìm nạp dữ liệu từ nơi khác. Các vùng chứa trợ giúp này duy trì mối quan hệ mật đới với vùng chứa ứng dụng của chúng tôi và do đó, cần giao tiếp trực tiếp với các vùng chứa ứng dụng và truy cập dữ liệu từ các vùng chứa đó. Để làm được điều này, chúng tôi cần duy trì bản đồ về những ứng dụng và vùng chứa trợ giúp nào được kết nối với nhau, chúng tôi sẽ cần tự thiết lập kết nối mạng giữa các vùng chứa này bằng cách sử dụng các liên kết và mạng tùy chỉnh, chúng tôi sẽ cần tạo các khối có thể chia sẻ và chia sẻ nó giữa các vùng chứa và duy trì một bản đồ về điều đó. Và quan trọng nhất là chúng ta cần theo dõi trạng thái của ứng dụng

container và khi nó chết, hãy tắt thủ công container trợ giúp cũng như nó không còn cần thiết nữa. Khi một vùng chứa mới được triển khai, chúng tôi cũng sẽ cần triển khai vùng chứa trợ giúp mới.

Với POD, kubernetes tự động thực hiện tất cả những điều này cho chúng ta. Chúng ta chỉ cần xác định những vùng chứa mà POD bao gồm và các vùng chứa trong POD theo mặc định sẽ có quyền truy cập vào cùng một bộ lưu trữ, cùng một không gian tên mạng và cùng số phận vì chúng sẽ được tạo và hủy cùng nhau.

Ngay cả khi ứng dụng của chúng tôi không quá phức tạp và chúng tôi có thể sử dụng một vùng chứa duy nhất, kubernetes vẫn yêu cầu bạn tạo POD. Nhưng điều này sẽ tốt về lâu dài vì ứng dụng của bạn hiện đã được trang bị cho những thay đổi về kiến trúc và quy mô trong tương lai.

Tuy nhiên, vùng chứa nhiều nhóm là một trường hợp hiếm khi sử dụng và chúng ta sẽ sử dụng một vùng chứa duy nhất cho mỗi POD trong khóa học này.

```
kubectl chạy nginx --image nginx
kubectl lấy nhóm
C:\Kubernetes>kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
nginx-8586cf59-whssr   0/1     ContainerCreating   0          3s

C:\Kubernetes>kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
nginx-8586cf59-whssr   1/1     Running    0          8s
```

Bây giờ chúng ta hãy xem cách triển khai POD. Trước đó chúng ta đã tìm hiểu về lệnh chạy kubectl. Lệnh này thực sự làm gì là nó triển khai một vùng chứa docker bằng cách tạo POD. Vì vậy, trước tiên, nó tự động tạo POD và triển khai một phiên bản của hình ảnh docker nginx. Nhưng liệu nó có lấy được hình ảnh ứng dụng từ đó không? Để làm được điều đó, bạn cần chỉ định tên hình ảnh bằng tham số `--image`. Hình ảnh ứng dụng, trong trường hợp này là hình ảnh nginx, được tải xuống từ kho lưu trữ trung tâm docker. Docker hub như chúng ta đã thảo luận là một kho lưu trữ công cộng, nơi lưu trữ các hình ảnh docker mới nhất của nhiều ứng dụng khác nhau. Bạn có thể định cấu hình kubernetes để lấy hình ảnh từ trung tâm docker công khai hoặc lưu trữ riêng trong tổ chức.

Bây giờ chúng ta đã tạo POD, làm cách nào để xem danh sách POD có sẵn? Lệnh kubectl get PODs giúp chúng ta xem danh sách các nhóm trong cụm của mình. Trong trường hợp này, chúng ta thấy pod đang ở trạng thái ContainerCreating và sớm chuyển sang trạng thái Running khi nó thực sự đang chạy.

Cũng nên nhớ rằng chúng ta chưa thực sự nói về các khái niệm về cách người dùng có thể truy cập máy chủ web nginx. Và vì vậy, ở trạng thái hiện tại, chúng tôi chưa làm cho người dùng bên ngoài có thể truy cập máy chủ web. Tuy nhiên, bạn có thể truy cập nội bộ từ Nút. Hiện tại, chúng ta sẽ chỉ xem cách triển khai POD và trong bài giảng sau khi chúng ta tìm hiểu

về mạng và dịch vụ, chúng ta sẽ biết cách giúp người dùng cuối có thể tiếp cận dịch vụ này.



Thử nghiệm

---

DƯỚI

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.

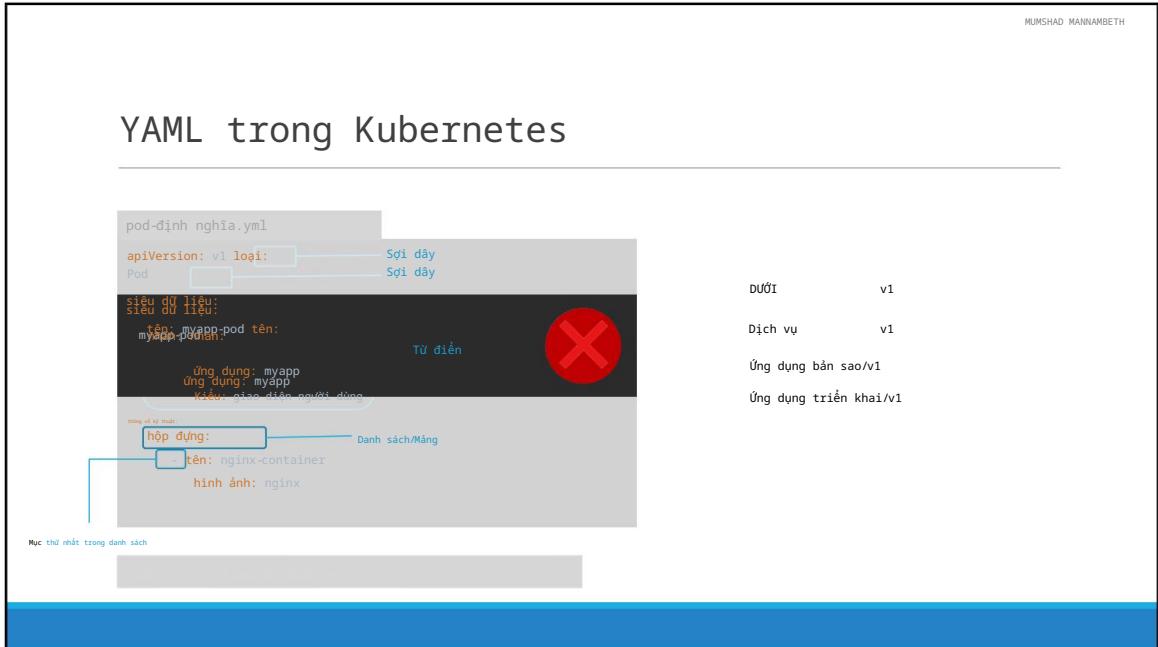
## POD

Với YAML



Xin chào và chào mừng bạn đến với bài giảng này. Trong bài giảng này, chúng ta sẽ nói về việc tạo POD bằng tệp cấu hình dựa trên YAML.

## YAML trong Kubernetes



Trong bài giảng trước chúng ta đã tìm hiểu về file YAML nói chung. Bây giờ chúng ta sẽ tìm hiểu cách phát triển các tệp YAML dành riêng cho Kubernetes. Kubernetes sử dụng các tệp YAML làm đầu vào để tạo các đối tượng như POD, Bản sao, Triển khai, Dịch vụ, v.v.

Tất cả đều theo cấu trúc tương tự. Tệp định nghĩa kubernetes luôn chứa 4 trường cấp cao nhất. `apiVersion`, `loại`, `siêu dữ liệu` và `thông số kỹ thuật`. Đây là các thuộc tính cấp cao nhất hoặc cấp gốc. Hãy coi họ như anh em ruột, con cái cùng một cha mẹ. Đây đều là các trường BẮT BUỘC, vì vậy bạn PHẢI có chúng trong tệp cấu hình của mình.

Chúng ta hãy nhìn vào từng người trong số họ. Cái đầu tiên là `apiVersion`. Đây là phiên bản API kubernetes mà chúng tôi đang sử dụng để tạo đối tượng. Tùy thuộc vào những gì chúng tôi đang cố gắng tạo ra, chúng tôi phải sử dụng `apiVersion` ĐÚNG. Hiện tại vì chúng tôi đang làm việc trên POD nên chúng tôi sẽ đặt `apiVersion` là `v1`. Một số giá trị có thể có khác cho trường này là `apps/v1beta1`, `Extensions/v1beta1`, v.v. Chúng ta sẽ xem những giá trị này dùng để làm gì ở phần sau của phần này. khóa học.

Tiếp theo là `loại`. Loại đề cập đến loại đối tượng mà chúng ta đang cố gắng tạo, trong trường hợp này là POD. Vì vậy chúng ta sẽ đặt nó là `Pod`. Một số giá trị khả thi khác ở đây có thể là `ReplicaSet` hoặc `Deployment` hoặc `Service`, đó là những gì bạn thấy trong trường loại ở bảng bên phải.

Tiếp theo là siêu dữ liệu. Siêu dữ liệu là dữ liệu về đối tượng như tên, nhãn, v.v. Như bạn có thể thấy, không giống như hai giá trị đầu tiên, bạn đã chỉ định một giá trị chuỗi, giá trị này ở dạng từ điển. Vì vậy, mọi thứ trong siêu dữ liệu đều được đặt ở bên phải một chút và vì vậy tên và nhãn là con của siêu dữ liệu. Số lượng khoảng trắng trước tên và nhãn của hai thuộc tính không quan trọng, nhưng chúng phải giống nhau vì chúng là anh em ruột. Trong trường hợp này, nhãn có nhiều khoảng trắng ở bên trái hơn tên và vì vậy bây giờ nó là con của thuộc tính tên thay vì anh chị em. Ngoài ra, hai thuộc tính phải có NHIỀU khoảng trắng hơn phần gốc của nó, đó là siêu dữ liệu, để nó nằm ở bên phải một chút. Trong trường hợp này, cả 3 đều có cùng số khoảng trắng phía trước và vì vậy họ đều là anh em ruột, điều này không đúng. Trong siêu dữ liệu, tên là giá trị chuỗi - vì vậy bạn có thể đặt tên cho POD myapp-pod của mình - và nhãn là từ điển. Vì vậy, nhãn là một từ điển nằm trong từ điển siêu dữ liệu. Và nó có thể có bất kỳ cặp khóa và giá trị nào bạn muốn. Hiện tại tôi đã thêm ứng dụng nhãn có giá trị myapp. Tương tự, bạn có thể thêm các nhãn khác khi thấy phù hợp, điều này sẽ giúp bạn xác định các đối tượng này sau này. Ví dụ: có 100 POD chạy giao diện người dùng

Ứng dụng và 100 trong số chúng đang chạy ứng dụng phụ trợ hoặc cơ sở dữ liệu, bạn sẽ KHÓ KHÓ để nhóm các POD này sau khi chúng được triển khai. Nếu bây giờ bạn gắn nhãn chúng là giao diện người dùng, mặt sau hoặc cơ sở dữ liệu, thì bạn sẽ có thể lọc POD dựa trên nhãn này sau này.

**Điều QUAN TRỌNG** cần lưu ý là trong siêu dữ liệu, bạn chỉ có thể chỉ định tên hoặc nhãn hoặc bất kỳ thứ gì khác mà Kubernetes mong đợi nằm trong siêu dữ liệu. Bạn KHÔNG THỂ thêm bất kỳ thuộc tính nào khác theo ý muốn của bạn. Tuy nhiên, dưới nhãn, bạn CÓ THỂ có bất kỳ loại cặp khóa hoặc giá trị nào mà bạn thấy phù hợp. Vì vậy, điều QUAN TRỌNG là phải hiểu từng tham số này mong đợi điều gì.

Cho đến nay, chúng tôi chỉ đề cập đến loại và tên của đối tượng mà chúng ta cần tạo, đó là POD có tên myapp-pod, nhưng chúng tôi chưa thực sự chỉ định vùng chứa hoặc hình ảnh mà chúng tôi cần trong nhóm. Phần cuối cùng trong tệp cấu hình là thông số kỹ thuật được viết dưới dạng spec. Tùy thuộc vào đối tượng mà chúng ta sẽ tạo, đây là lúc chúng ta cung cấp thông tin bổ sung cho Kubernetes liên quan đến đối tượng đó. Điều này sẽ khác nhau đối với các đối tượng khác nhau, vì vậy điều quan trọng là phải hiểu hoặc tham khảo phần tài liệu để có định dạng phù hợp cho từng đối tượng. Vì chúng ta chỉ tạo một nhóm với một thùng chứa duy nhất nên việc này rất dễ dàng. Spec là một từ điển nên hãy thêm một thuộc tính bên dưới nó gọi là vùng chứa, là một danh sách hoặc một mảng. Lý do thuộc tính này là một danh sách là vì POD có thể có nhiều vùng chứa bên trong chúng như chúng ta đã học trong bài giảng trước đó. Tuy nhiên, trong trường hợp này, chúng tôi sẽ chỉ thêm một mục duy nhất vào danh sách vì chúng tôi dự định chỉ có một vùng chứa duy nhất trong POD. Mục trong danh sách là từ điển, vì vậy hãy thêm thuộc tính tên và hình ảnh là nginx.

Sau khi tệp được tạo, hãy chạy lệnh `kubectl create -f` theo sau là tên tệp là `pod-def định.yaml` và kubernetes sẽ tạo nhóm.

Vì vậy, để tóm tắt hãy nhớ 4 thuộc tính cấp cao nhất. `apiVersion`, `loại`, `siêu dữ liệu` và `thông số kỹ thuật`. Sau đó bắt đầu bằng cách thêm các giá trị vào các giá trị đó tùy thuộc vào đối tượng bạn đang tạo.

# Lệnh

```
> kubectl nhận nhóm
TÊN      SẴN SÀNG   TRẠNG THÁI     TÌM KIẾM
myapp-pod  1/1       Đang chạy  8

> kubectl mô tả pod myapp-pod
Tên: myapp-pod
Không gian tên: mặc định
Nhóm: default
Thời gian bắt đầu: Thứ bảy, 03/03/2018 14:26:14 +0800
Nhóm: ứng dụng myapp
Chủ thính: none*
Tình trạng: Đang chạy
IP: 10.244.8.24
Hỗn hợp:
ID vipec chứa: docker://03bb66c8c42a60d8b/Nw9cf1408faw1lc38663e4918de2f5a783e7688dc9d
Hình ảnh: nginx
ID hinh ảnh: docker/gullable/nginx@sha256:4771bd95978c7da6529e11803ew1da2592f5ea2618d23effa7a4cab1ce5de
màn hình:
Tình trạng: Đang chạy
Giờ bắt đầu: Thứ bảy, 03/03/2018 14:26:21 +0800
Sẵn sàng:
Số lượng lõi: 8
Môi trường: không có-
Gắn kết:
    /var/run/secrets/kubernetes.io/serviceaccount từ default-token-v95wz (x)
Số liệu:
Loại: Trạng thái
đã khởi tạo: Đang chạy
sẵn sàng: Đang chạy
Đang chạy: Không có
Sự kiện:
Loại lý do ..... Tuổi tử Tin nhắn
Thay đổi trạng thái thường: Tính lập lịch mặc định 34s Đã gán thành công myapp-pod cho minikube
Bình thường Thành côngMountVolume 33s kubelet, minikube MountVolume_Setup đã thành công cho tập "default-token-v95wz".
Bình thường Kéo Bình thường: kubelet 33s, hình ảnh kéo minikube "nginx"
Thường Kéo Bình thường: 27s kubelet, minikube Đã kéo thành công hình ảnh "nginx"
Thường Kéo Bình thường: 27s kubelet, minikube Đã bắt đầu container
Thường bắt đầu
```

Khi chúng tôi tạo nhóm, bạn thấy nó như thế nào? Sử dụng lệnh kubectl get pod để xem danh sách các pod có sẵn. Trong trường hợp này nó chỉ là một. Để xem thông tin chi tiết về nhóm, hãy chạy lệnh kubectl mô tả nhóm. Điều này sẽ cho bạn biết thông tin về POD, thời điểm nó được tạo, nhãn nào được gán cho nó, vùng chứa docker nào là phần của nó và các sự kiện liên quan đến POD đó.



## Thử nghiệm

---

POD sử dụng YAML

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.



## Tips & Tricks

Lời khuyên &  
Thủ thuật

WORKING WITH PPTES

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.

## Tài nguyên

---

Liên kết đến các phiên bản và nhóm- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.9/#replicaset-v1-apps>

<https://plugins.jetbrains.com/plugin/9354-kubernetes-and-openshift-resource-support>

## Replication Controller

Bộ điều khiển sao chép



mumshad.mannambeth

Xin chào và chào mừng bạn đến với bài giảng này về Bộ điều khiển Kubernetes. Trong bài giảng này chúng ta sẽ thảo luận về Kubernetes Controllers. Bộ điều khiển là bộ não đằng sau Kubernetes.

Chúng là các quá trình giám sát các đối tượng kubernetes và phản hồi tương ứng. Trong bài giảng này chúng ta sẽ thảo luận cụ thể về một bộ điều khiển. Và đó là Bộ điều khiển sao chép.

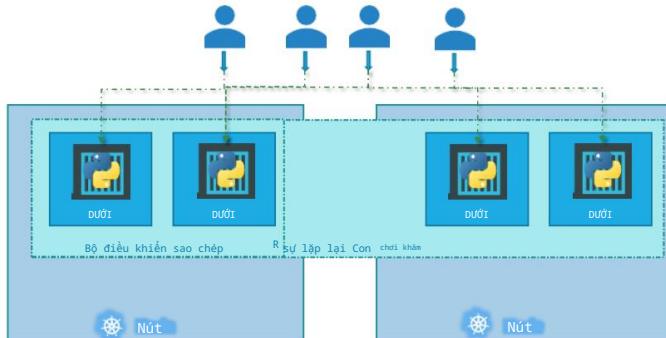
## Tính sẵn sàng cao



Vậy bản sao là gì và tại sao chúng ta cần bộ điều khiển sao chép? Hãy quay lại kịch bản đầu tiên khi chúng ta có một POD duy nhất chạy ứng dụng của mình. Điều gì sẽ xảy ra nếu vì lý do nào đó, ứng dụng của chúng ta gặp sự cố và POD bị lỗi? Người dùng sẽ không thể truy cập ứng dụng của chúng tôi nữa. Để ngăn người dùng mất quyền truy cập vào ứng dụng của chúng tôi, chúng tôi muốn có nhiều phiên bản hoặc POD chạy cùng lúc. Bằng cách đó, nếu một cái bị lỗi, chúng tôi vẫn có ứng dụng của mình chạy trên cái kia. Bộ điều khiển sao chép giúp chúng tôi chạy nhiều phiên bản của một POD duy nhất trong cụm Kubernetes, do đó mang lại Tính sẵn sàng cao.

Vậy điều đó có nghĩa là bạn không thể sử dụng bộ điều khiển sao chép nếu bạn dự định có một POD duy nhất? KHÔNG! Ngay cả khi bạn có một POD duy nhất, bộ điều khiển sao chép có thể trợ giúp bằng cách tự động đưa ra một POD mới khi POD hiện có bị lỗi. Do đó, bộ điều khiển sao chép đảm bảo rằng số lượng POD được chỉ định luôn chạy. Dù chỉ là 1 hay 100.

## Cân bằng tải và mở rộng quy mô



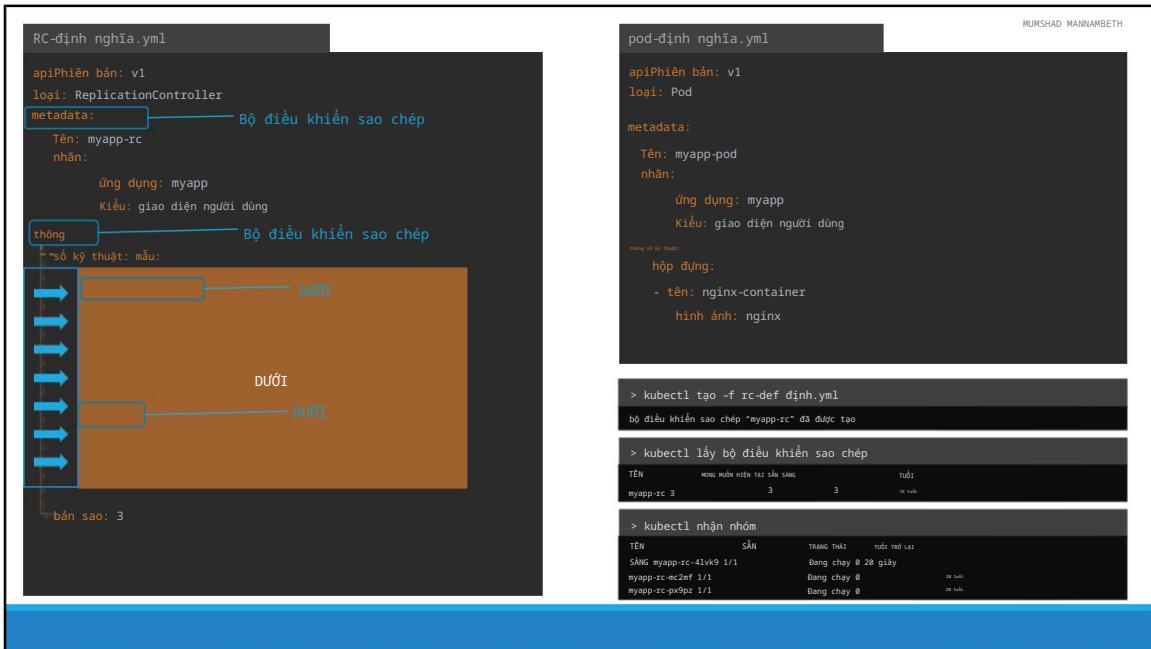
Một lý do khác khiến chúng ta cần bộ điều khiển sao chép là tạo nhiều POD để chia sẻ tải trên chúng. Ví dụ: trong kịch bản đơn giản này, chúng tôi có một POD duy nhất phục vụ một nhóm người dùng. Khi số lượng người dùng tăng lên, chúng tôi triển khai thêm POD để cân bằng tải trên hai nhóm. Nếu nhu cầu tăng thêm và nếu chúng tôi hết tài nguyên trên nút đầu tiên, chúng tôi có thể triển khai các POD bổ sung trên các nút khác trong cụm. Như bạn có thể thấy, bộ điều khiển sao chép trải rộng trên nhiều nút trong cụm. Nó giúp chúng tôi cân bằng tải trên nhiều nhóm trên các nút khác nhau cũng như mở rộng quy mô ứng dụng của chúng tôi khi nhu cầu tăng lên.

Bộ điều khiển sao chép

Bộ bản sao

Điều quan trọng cần lưu ý là có hai thuật ngữ tương tự. Bộ điều khiển sao chép và bộ bản sao. Cả hai đều có cùng mục đích nhưng chúng không giống nhau. Bộ điều khiển sao chép là công nghệ cũ hơn đang được thay thế bởi Bộ bản sao. Bộ bản sao là cách mới được đề xuất để thiết lập bản sao. Tuy nhiên, bất cứ điều gì chúng ta đã thảo luận trong một số trang trình bày trước đó vẫn có thể áp dụng được cho cả hai công nghệ này. Có những khác biệt nhỏ trong cách mỗi hoạt động và chúng ta sẽ xem xét điều đó một chút.

Vì vậy, chúng tôi sẽ cố gắng tuân thủ Bộ bản sao trong tất cả các bản demo và triển khai của mình tiến về phía trước.



Bây giờ chúng ta hãy xem cách chúng ta tạo một bộ điều khiển sao chép. Giống như bài giảng trước, chúng ta bắt đầu bằng cách tạo một tệp định nghĩa bộ điều khiển sao chép. Chúng tôi sẽ đặt tên nó là định nghĩa RC.yml. Như với bất kỳ tệp định nghĩa Kubernetes nào, chúng ta sẽ có 4 phần. Các apiVersion, loại, siêu dữ liệu và thông số kỹ thuật. apiVersion dành riêng cho những gì chúng tôi đang tạo. Trong trường hợp này, bộ điều khiển sao chép được hỗ trợ trong Kubernetes apiVersion v1. Vì vậy, chúng tôi sẽ viết nó là v1. Loại mà chúng ta biết là ReplicationController. Trong siêu dữ liệu, chúng tôi sẽ thêm tên và gọi nó là myapp-rc. Và chúng tôi cũng sẽ thêm một số nhãn, ứng dụng, loại và gán giá trị cho chúng. Cho đến nay, nó rất giống với cách chúng ta tạo POD trong phần trước. Phần tiếp theo là phần quan trọng nhất của tệp định nghĩa và đó là thông số kỹ thuật được viết dưới dạng spec. Đôi với mọi Kubernetes định nghĩa, phần thông số xác định những gì bên trong đối tượng chúng ta đang tạo. Trong trường hợp này, chúng ta biết rằng bộ điều khiển sao chép tạo ra nhiều phiên bản của POD. Nhưng POD là gì? Chúng tôi tạo một phần mẫu theo thông số kỹ thuật để cung cấp mẫu POD được bộ điều khiển sao chép sử dụng để tạo bản sao. Bây giờ làm cách nào để XÁC ĐỊNH mẫu POD? Nó không khó lắm vì chúng ta đã làm điều đó ở bài tập trước. Hãy nhớ rằng chúng ta đã tạo một tệp định nghĩa nhóm trong bài tập trước. Chúng tôi có thể sử dụng lại nội dung của cùng một tệp để điền vào phần mẫu. Di chuyển tất cả nội dung của tệp định nghĩa nhóm vào phần mẫu của bộ điều khiển sao chép, ngoại trừ hai dòng đầu tiên - đó là apiVersion và

loại. Hãy nhớ bất cứ điều gì chúng ta di chuyển phải ở DƯỚI phần mẫu. Có nghĩa là, chúng phải được đặt ở bên phải và có nhiều khoảng trống trước chúng hơn chính dòng mẫu. Nhìn vào tệp của chúng tôi, hiện tại chúng tôi có hai phần siêu dữ liệu - một phần dành cho Bộ điều khiển sao chép và một phần khác dành cho POD và chúng tôi có hai phần thông số kỹ thuật - một phần dành cho mỗi phần. Chúng tôi đã lồng hai tệp định nghĩa vào nhau. Bộ điều khiển sao chép là cha mẹ và định nghĩa nhóm là con.

Bây giờ, vẫn còn thiếu một cái gì đó. Chúng tôi chưa đề cập đến số lượng bản sao chúng tôi cần trong bộ điều khiển sao chép. Để làm được điều đó, hãy thêm một thuộc tính khác vào thông số kỹ thuật được gọi là bản sao và nhập số lượng bản sao bạn cần theo thuộc tính đó. Hãy nhớ rằng mẫu và bản sao là con trực tiếp của phần thông số kỹ thuật. Vì vậy, họ là anh em ruột và phải nằm trên cùng một đường thẳng đứng: có số khoảng trống phía trước bằng nhau.

Khi tệp đã sẵn sàng, hãy chạy lệnh tạo kubectl và nhập tệp bằng tham số -f. Bộ điều khiển sao chép được tạo. Khi bộ điều khiển sao chép được tạo, trước tiên, nó sẽ tạo các POD bằng cách sử dụng mẫu định nghĩa nhóm với số lượng tùy ý, trong trường hợp này là 3. Để xem danh sách các bộ điều khiển sao chép đã tạo, hãy chạy lệnh kubectl get sao chép bộ điều khiển và bạn sẽ thấy bộ điều khiển sao chép được liệt kê. Chúng ta cũng có thể xem số lượng bản sao hoặc nhóm mong muốn, số lượng bản sao hiện tại và số lượng bản sao đã sẵn sàng. Nếu bạn muốn xem các nhóm được tạo bởi bộ điều khiển sao chép, hãy chạy lệnh kubectl get pod và bạn sẽ thấy 3 nhóm đang chạy. Lưu ý rằng tất cả chúng đều bắt đầu bằng tên của bộ điều khiển sao chép là myapp-rc cho biết rằng tất cả chúng đều được bộ điều khiển sao chép tạo tự động.

bản sao-dịnh nghĩa.yml

```
apiVersion: Ứng dụng/v1
loại: Bản sao
siêu dữ liệu:
  tên: myapp-replicaset
nhân:
  Ứng dụng: myapp
  Kiểu: giao diện người dùng
mô tả kỹ thuật:
  bản mẫu:
```

DƯỚI

bản sao: 3
bộ chọn:
 matchLabels:
 Kiểu: giao diện người dùng

lỗi: Không thể nhận ra "replicaSetsdeatt-a": định nghĩa.yml": không khớp với /,naKmien:d=mRyeapplpi-pcoadSet

pod-dịnh nghĩa.yml

```
apiPhiên bản: v1
loại: Pod
nhân:
  Ứng dụng: myapp
  Kiểu: giao diện người dùng
mô tả kỹ thuật:
  hộp đựng:
    - tên: nginx-container
    hình ảnh: nginx
```

```
> kubectl create -f replicaset-def định.yml
bản sao "myapp-replicaset" đã được tạo
```

```
> kubectl lấy bản sao
TÊN           MONG MUỐN HIỆN TẠI myapp-replicaset-0   SẴN SÀNG   TUỔI
replicaset-0   Đang chạy 0                                10 ngày
```

```
> kubectl nhận nhóm
TÊN           SẴN SÀNG   TRẠM THÁI   TUỔI HỎ LỐI
SANG myapp-replicaset-0dd19 1/1 myapp-replicaset-0jtpx 1/1 myapp-replicaset-hg84n 1/1   Đang chạy 0   10 ngày
```

Những gì chúng ta vừa thấy là ReplicationController. Bây giờ chúng ta hãy xem ReplicaSet. Nó rất giống với bộ điều khiển sao chép. Như thường lệ, đầu tiên chúng ta có apiVersion, loại, siêu dữ liệu và thông số kỹ thuật. Tuy nhiên, apiVersion hơi khác một chút. Đó là apps/v1 khác với những gì chúng tôi có trước đây cho bộ điều khiển sao chép. Đối với bộ điều khiển sao chép, nó chỉ đơn giản là v1. Nếu bạn hiểu sai điều này, bạn có thể gặp lỗi giống như thế này. Nó sẽ nói rằng không phù hợp với loại ReplicaSet, vì phiên bản api kubernetes được chỉ định không hỗ trợ cho ReplicaSet.

Loại này sẽ là ReplicaSet và chúng tôi thêm tên và nhãn vào siêu dữ liệu.

Phần đặc tả trông rất giống với bộ điều khiển sao chép. Nó có một phần mẫu mà chúng tôi đã cung cấp định nghĩa nhóm như trước đây. Vì vậy, tôi sẽ sao chép nội dung từ tệp định nghĩa nhóm. Và chúng tôi đặt số lượng bản sao là 3. Tuy nhiên, có một điểm khác biệt lớn giữa bộ điều khiển sao chép và bộ bản sao. Bộ bản sao yêu cầu định nghĩa bộ chọn. Phần bộ chọn giúp bản sao xác định nhóm nào nằm trong đó. Nhưng tại sao bạn phải chỉ định POD nào thuộc nó, nếu bạn đã cung cấp nội dung của chính tệp định nghĩa nhóm trong mẫu? Đó là VÌ, bộ bản sao CŨNG có thể quản lý các nhóm không được tạo như một phần của

tạo bản sao. Ví dụ: có những nhóm được tạo TRƯỚC KHI tạo Bản sao khớp với các nhãn được chỉ định trong bộ chọn, bộ bản sao cũng sẽ xem xét các nhóm Đó khi tạo bản sao. Tôi sẽ giải thích điều này trong slide tiếp theo.

Nhưng trước khi chúng ta đi sâu vào vấn đề đó, tôi muốn đề cập rằng bộ chọn là một trong những điểm khác biệt chính giữa bộ điều khiển sao chép và bộ bản sao. Bộ chọn không phải là trường BẤT BUỘC trong trường hợp bộ điều khiển sao chép, nhưng nó vẫn có sẵn. Khi bạn bỏ qua nó, như chúng ta đã làm trong trang trình bày trước, nó sẽ coi nó giống với các nhãn được cung cấp trong tệp định nghĩa nhóm. Trong trường hợp thiết lập bản sao, IS yêu cầu đầu vào của người dùng cho thuộc tính này. Và nó phải được viết dưới dạng matchLabels như được hiển thị ở đây. Bộ chọn matchLabels chỉ cần khớp các nhãn được chỉ định bên dưới nó với các nhãn trên POD. Bộ chọn bản sao cũng cung cấp nhiều tùy chọn khác để khớp các nhãn không có sẵn trong bộ điều khiển sao chép.

Và như thường lệ, để tạo một ReplicaSet, hãy chạy lệnh tạo kubectl cung cấp tệp định nghĩa làm đầu vào và để xem các bản sao đã tạo, hãy chạy lệnh kubectl get replicaset. Để lấy danh sách các nhóm, chỉ cần chạy lệnh kubectl get pod.

## Nhãn và Bộ chọn



Vậy vấn đề với Nhãn và Bộ chọn là gì? Tại sao chúng ta gắn nhãn POD và đối tượng của mình trong kubernetes? Chúng ta hãy xem xét một kịch bản đơn giản. Giả sử chúng tôi đã triển khai 3 phiên bản ứng dụng web giao diện người dùng của mình dưới dạng 3 POD. Chúng tôi muốn tạo bộ điều khiển sao chép hoặc bộ bản sao để đảm bảo rằng chúng tôi có 3 POD hoạt động bất cứ lúc nào. Và CÓ, đó là một trong những trường hợp sử dụng bộ bản sao. Bạn CÓ THỂ sử dụng nó để giám sát các nhóm hiện có, nếu bạn đã tạo chúng, như trong ví dụ này. Trong trường hợp chúng chưa được tạo, bộ bản sao sẽ tạo chúng cho bạn. Vai trò của bản sao là giám sát các nhóm và nếu bất kỳ nhóm nào bị lỗi, hãy triển khai các nhóm mới. Bộ bản sao thực tế là một quy trình giám sát các nhóm. Nay giờ, làm cách nào để bản sao BIỆT nhóm nào cần giám sát.

Có thể có hàng trăm POD khác trong cụm chạy ứng dụng khác. Đây là việc dán nhãn POD của chúng tôi trong quá trình tạo sẽ rất hữu ích. Nay giờ chúng tôi có thể cung cấp các nhãn này làm bộ lọc cho bản sao. Trong phần bộ chọn, chúng tôi sử dụng bộ lọc matchLabels và cung cấp cùng một nhãn mà chúng tôi đã sử dụng khi tạo nhóm. Bằng cách này, bản sao sẽ biết nhóm nào cần giám sát.

bản sao-định nghĩa.yml

```

apiVersion: Ứng dụng/v1
loại: Bản sao
metadata:
  Tên: myapp-bản sao
  nhãn:
    ứng dụng: myapp
    loại: giao diện người dùng

thông số kỹ thuật:
  bản mẫu:
    metadata:
      Tên: myapp-pod
      nhãn: ứng
      dung: myapp
      loại: từ etm pe ln ad te

    thông số kỹ thuật:
      hộp dung:
        - tên: nginx-container
        hình ảnh: nginx

  bản sao: 3
  bộ chọn:
    matchLabels:
      Kiểu: giao diện người dùng

```

Bây giờ hãy để tôi hỏi bạn một câu hỏi tương tự. Trong phần đặc tả bản sao, chúng ta đã biết rằng có 3 phần: Mẫu, bản sao và bộ chọn. Chúng tôi cần 3 bản sao và chúng tôi đã cập nhật bộ chọn dựa trên cuộc thảo luận của chúng tôi trong trang trình bày trước. Ví dụ: chúng ta có kịch bản tương tự như trong slide trước đó là chúng ta đã có 3 POD hiện có đã được tạo và chúng ta cần tạo một bộ bản sao để giám sát các POD nhằm đảm bảo luôn có tối thiểu 3 POD đang chạy. Khi bộ điều khiển sao chép được tạo, nó sẽ KHÔNG triển khai một phiên bản POD mới vì 3 trong số chúng có nhãn phù hợp đã được tạo. Trong trường hợp đó, chúng tôi có thực sự cần cung cấp phần mẫu trong đặc tả bộ bản sao không, vì chúng tôi không mong đợi bộ bản sao sẽ tạo POD mới khi triển khai? Có, chúng tôi làm như vậy, Vì trong trường hợp một trong các POD bị lỗi trong tương lai, bộ bản sao cần tạo một POD mới để duy trì số lượng POD mong muốn. Và để bộ bản sao tạo POD mới, cần phải có phần định nghĩa mẫu.

# Tỉ lệ

```

bản sao-dịnh nghĩa.yml
MUMSHAD MANNAMBETH

apiVersion: loại ứng dụng/
v1 : Siêu dữ liệu bản
sao:
  Tên: myapp-bản sao
  nhãn:
    ứng dụng: myapp
    Kiểu: giao diện người dùng
    thông số kỹ thuật:
      mẫu: siêu dữ
      liệu:
        Tên: myapp-pod
        nhãn:
          ứng dụng: myapp
          Kiểu: giao diện người dùng
          thông số kỹ thuật:
            hộp đựng:
              - tên: nginx-container
                hình ảnh: nginx
    bản sao: 30
    bộ chọn:
      matchLabels:
        Kiểu: giao diện người dùng

```

Hãy xem cách chúng tôi mở rộng quy mô bản sao. Giả sử chúng tôi bắt đầu với 3 bản sao và trong tương lai chúng tôi quyết định mở rộng quy mô lên 6. Làm cách nào để cập nhật bộ bản sao của mình để mở rộng quy mô thành 6 bản sao. Vâng, có nhiều cách để làm điều đó. Đầu tiên là cập nhật số lượng bản sao trong tệp định nghĩa lên 6. Sau đó chạy lệnh kubectl thay thế chỉ định cùng một tệp bằng cách sử dụng tham số `-f` và điều đó sẽ cập nhật bộ bản sao để có 6 bản sao.

Cách thứ hai để làm điều đó là chạy lệnh thang đo kubectl. Sử dụng tham số bản sao để cung cấp số lượng bản sao mới và chỉ định cùng một tệp làm đầu vào. Bạn có thể nhập tệp định nghĩa hoặc cung cấp tên bản sao ở định dạng Tên LOẠI. Tuy nhiên, hãy nhớ rằng việc sử dụng tên tệp làm đầu vào sẽ không dẫn đến số lượng bản sao được cập nhật tự động trong tệp. Nói cách khác, số lượng bản sao trong tệp định nghĩa bản sao sẽ vẫn là 3 ngay cả khi bạn đã chia tỷ lệ bộ bản sao của mình để có 6 bản sao bằng cách sử dụng lệnh kubectlscale và tệp làm đầu vào.

Ngoài ra còn có các tùy chọn để tự động mở rộng quy mô bản sao dựa trên tải, nhưng đó là chủ đề nâng cao và chúng ta sẽ thảo luận về nó sau.

## lệnh

```
> kubectl create -f replicaset-def định.yml  
> kubectl lấy bản sao  
> kubectl xóa bản sao myapp-replicaset *Cũng xóa tất cả các POD cơ bản  
> kubectl thay thế -f replicaset-def định.yml  
> thang do kubectl -replicas=6 -f replicaset-def định.yml
```

Bây giờ chúng ta hãy xem lại các lệnh thật nhanh. Lệnh tạo kubectl, như chúng ta biết, được sử dụng để tạo một bộ thay thế. Bạn phải cung cấp tệp đầu vào bằng tham số -f. Sử dụng lệnh kubectl get để xem danh sách các bản sao được tạo. Sử dụng lệnh kubectl delete replicaset theo sau là tên của bộ bản sao để xóa bộ bản sao. Sau đó, chúng ta có lệnh kubectl thay thế để thay thế hoặc cập nhật bản sao và cả lệnh kubectl scale để chia tỷ lệ các bản sao chỉ từ dòng lệnh mà không cần phải sửa đổi tệp.



## Thử nghiệm

---

Bộ bản sao

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.

## Người giới thiệu

Bản sao được đặt làm mục tiêu tự động chia tỷ lệ nhóm ngang

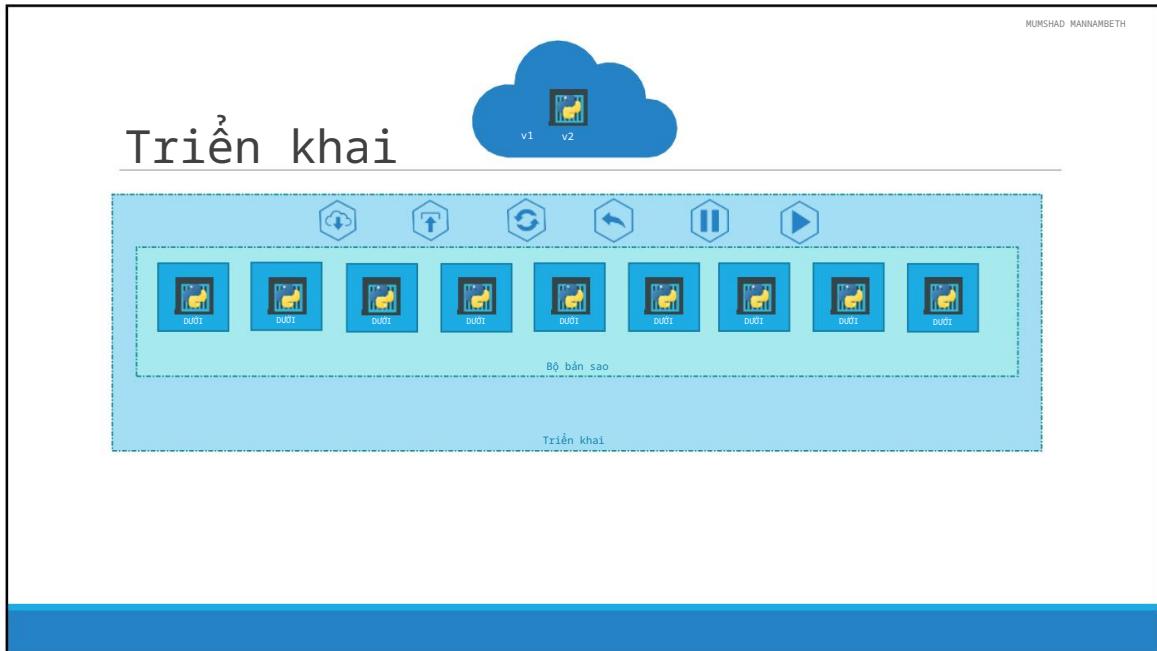
[https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/#replicaset-as-an\\_Horizontal-pod-autoscaler-target](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/#replicaset-as-an_Horizontal-pod-autoscaler-target)

## Deployment Triển khai



mumshad.mannambeth

Trong bài giảng này, chúng ta sẽ thảo luận về Triển khai Kubernetes.



Trong một phút, chúng ta hãy quên đi POD, bản sao cũng như các khái niệm kubernetes khác và nói về cách bạn có thể muốn triển khai ứng dụng của mình trong môi trường sản xuất. Ví dụ: bạn có một máy chủ web cần được triển khai trong môi trường sản xuất. Bạn không cần MỘT, mà cần nhiều trường hợp máy chủ web chạy vì những lý do rõ ràng.

Thứ hai, khi các phiên bản mới hơn của bản dựng ứng dụng có sẵn trên sổ đăng ký docker, bạn muốn NÂNG CẤP các phiên bản docker của mình một cách liền mạch.

Tuy nhiên, khi nâng cấp phiên bản của mình, bạn không muốn nâng cấp tất cả chúng cùng một lúc như chúng tôi vừa làm. Điều này có thể ảnh hưởng đến việc người dùng truy cập các ứng dụng của chúng tôi, vì vậy bạn có thể muốn nâng cấp từng ứng dụng một. Và loại nâng cấp đó được gọi là Cập nhật cuộn.

Giả sử một trong những nâng cấp bạn thực hiện dẫn đến lỗi không mong muốn và bạn được yêu cầu hoàn tác cập nhật gần đây. Bạn muốn có thể khôi phục lại những thay đổi đã được thực hiện gần đây.

Cuối cùng, ví dụ: bạn muốn thực hiện nhiều thay đổi đối với môi trường của mình

chẳng hạn như nâng cấp các phiên bản WebServer cơ bản, cũng như mở rộng quy mô môi trường của bạn và sửa đổi phân bổ tài nguyên, v.v. Bạn không muốn áp dụng từng thay đổi ngay sau khi lệnh được chạy, thay vào đó, bạn muốn áp dụng tạm dừng cho môi trường của mình, hãy thực hiện các thay đổi rồi tiếp tục để tất cả các thay đổi được triển khai cùng nhau.

Tất cả những khả năng này đều có sẵn với Triển khai kubernetes.

Cho đến nay trong khóa học này, chúng ta đã thảo luận về POD, triển khai các phiên bản đơn lẻ của ứng dụng của chúng ta, chẳng hạn như ứng dụng web trong trường hợp này. Mỗi container được đóng gói trong POD. Nhiều POD như vậy được triển khai bằng Bộ điều khiển sao chép hoặc Bộ bản sao. Và sau đó là Triển khai, một đối tượng kubernetes cao hơn trong hệ thống phân cấp. Việc triển khai cung cấp cho chúng tôi khả năng nâng cấp các phiên bản cơ bản một cách liền mạch bằng cách sử dụng các bản cập nhật luân phiên, hoàn tác các thay đổi cũng như tạm dừng và tiếp tục các thay đổi đối với hoạt động triển khai.

```

Sự định nghĩa

> kubectl tạo -f triển khai-dịnh nghĩa.yml
tríển khai myapp "tríển khai myapp" đã được tạo

> k xin hãy nhận triển khai
TÊN           MONG MUỐN HIỂN TẠI SẴN SÀNG   TUỔI
tríển khai 3          3                      3          2m
tríển khai 3          3                      3          2m
tríển khai 3          3                      3          2m

> k hãy lấy bản sao
TÊN           SẴN SÀNG   TRẠNG THÁI   TUỔI TRỞ LẠI
tríển khai-6795844b58-5rbj1l 1/1 myapp -tríển   Đang chạy 0   2m
khai-6795844b58-hd455 1/1 -tríển khai-6795844b58-1fjhv   Đang chạy 0   2m
tríển khai 1/1          Đang chạy 0   2m

> k xin hãy lấy vỏ
TÊN           SẴN SÀNG   TRẠNG THÁI   TUỔI TRỞ LẠI
tríển khai-6795844b58-5rbj1l 1/1 myapp -tríển   Đang chạy 0   2m
khai-6795844b58-hd455 1/1 -tríển khai-6795844b58-1fjhv   Đang chạy 0   2m
tríển khai 1/1          Đang chạy 0   2m

```

tríển khai-dịnh nghĩa.yml

apiVersion: Ứng dụng/v1

loại: Bản sao

metadata:

- Tên: triển khai myapp
- Nhân:

Ứng dụng: myapp

Kiểu: giao diện người dùng

thông số kỹ thuật:

bản mẫu:

metadata:

- Tên: myapp-pod
- Nhân:

Ứng dụng: myapp

Kiểu: giao diện người dùng

thông số kỹ thuật:

hộp ứng:

- tên: nginx-container
- hình ảnh: nginx

bản sao: 3

bộ chọn:

matchLabels:

Kiểu: giao diện người dùng

Vậy làm thế nào để chúng ta tạo một tệp triển khai. Giống như các thành phần trước, trước tiên chúng tôi tạo tệp định nghĩa triển khai. Nội dung của tệp định nghĩa triển khai hoàn toàn giống với tệp định nghĩa bản sao, ngoại trừ loại hiện sẽ là Triển khai.

Nếu chúng ta xem qua nội dung của tệp thì nó có apiVersion là apps/v1, siêu dữ liệu có tên và nhãn cũng như thông số kỹ thuật có mẫu, bản sao và bộ chọn. Mẫu có định nghĩa POD bên trong nó.

Khi tệp đã sẵn sàng, hãy chạy lệnh tạo kubectl và chỉ định tệp định nghĩa triển khai. Sau đó chạy lệnh kubectl get triển khai để xem quá trình triển khai mới được tạo. Việc triển khai sẽ tự động tạo một bộ bản sao. Vì vậy, nếu chạy lệnh kubectl get replicaset, bạn sẽ có thể thấy một bản sao mới trong tên của quá trình triển khai. Các bản sao cuối cùng sẽ tạo ra các nhóm, vì vậy nếu chạy lệnh kubectl get pod, bạn sẽ có thể thấy các nhóm có tên triển khai và bản sao.

Cho đến nay không có nhiều sự khác biệt giữa bản sao và bản triển khai, ngoại trừ thực tế là việc triển khai đã tạo ra một đối tượng Kubernetes mới được gọi là

triển khai. Chúng ta sẽ xem cách tận dụng lợi thế của việc triển khai bằng cách sử dụng các trường hợp sử dụng mà chúng ta đã thảo luận ở trang trình bày trước trong các bài giảng sắp tới.

# lệnh

```
> kubectl có được tắt cả
          MONG MUỐN CẤP NHẬT HIỆN TẠI          TUỔI CÓ SẴN
TÊN                                         3           3           3           9h
triển khai/triển khai myapp 3

          MONG MUỐN HOÀN TỐI SẴN SÀNG          TUỔI
TÊN                                         3           3           9h
rs/myapp-triển khai-6795844b58 3

          SẴN SÀNG          TRẠNG THÁI          TUỔI THỎ LAI
TÊN                                         3           Đang chạy 0           9h
po/myapp-deployment-6795844b58-5rbj1 1/1 po/myapp-
deployment-6795844b58-h4w55 1/1 po/myapp-deployment-6795844b58-
ifjhv 1/1                                         Đang chạy 0           9h
                                         Đang chạy 0           9h
```

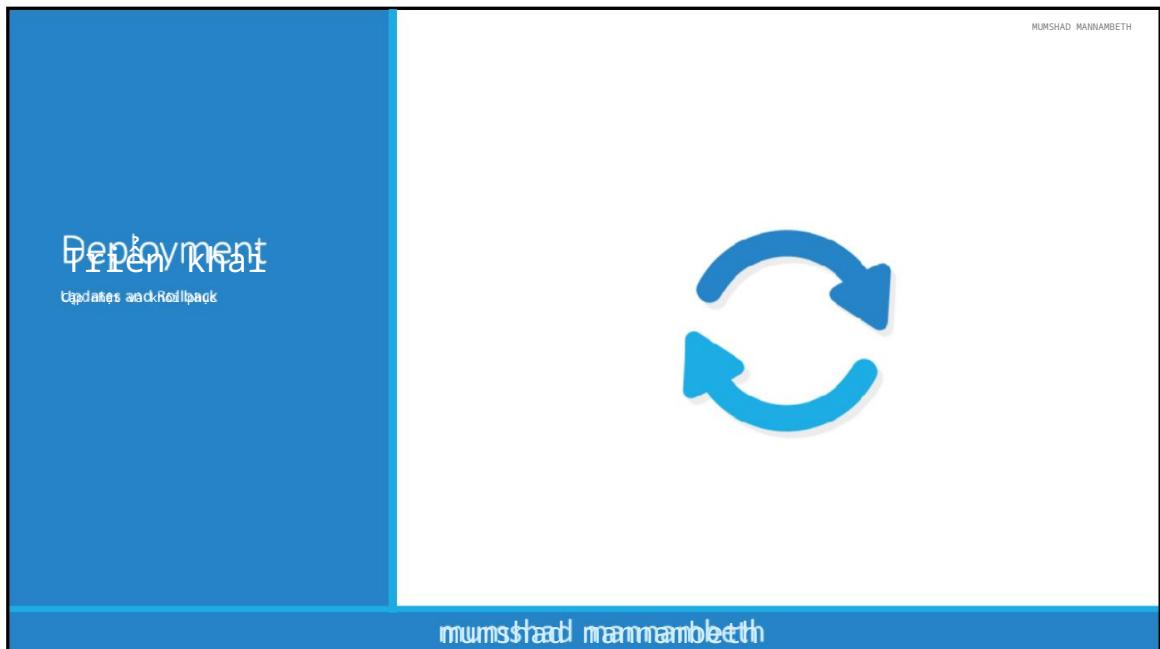
Để xem tất cả các đối tượng đã tạo cùng một lúc, hãy chạy lệnh kubectl get all.



## Thử nghiệm

Triển khai

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.



Trong bài giảng này, chúng ta sẽ nói về các bản cập nhật và khôi phục trong quá trình Triển khai.

## Triển khai và phiên bản



Trước khi xem cách chúng tôi nâng cấp ứng dụng của mình, hãy cố gắng tìm hiểu Triển khai và Lập phiên bản trong quá trình triển khai. Bất cứ khi nào bạn tạo một triển khai mới hoặc nâng cấp hình ảnh trong một triển khai hiện có, nó sẽ kích hoạt Triển khai. Triển khai là quá trình triển khai hoặc nâng cấp dần dần vùng chứa ứng dụng của bạn. Khi bạn tạo một triển khai lần đầu tiên, nó sẽ kích hoạt quá trình triển khai. Một đợt triển khai mới sẽ tạo ra một bản sửa đổi Triển khai mới. Hãy gọi nó là bản sửa đổi 1. Trong tương lai khi ứng dụng được nâng cấp - nghĩa là khi phiên bản vùng chứa được cập nhật lên phiên bản mới - một bản giới thiệu mới sẽ được kích hoạt và một bản sửa đổi triển khai mới được tạo có tên là Bản sửa đổi 2. Điều này giúp chúng tôi theo dõi những thay đổi được thực hiện đối với quá trình triển khai của chúng tôi và cho phép chúng tôi quay lại phiên bản triển khai trước đó nếu cần.

## Lệnh triển khai

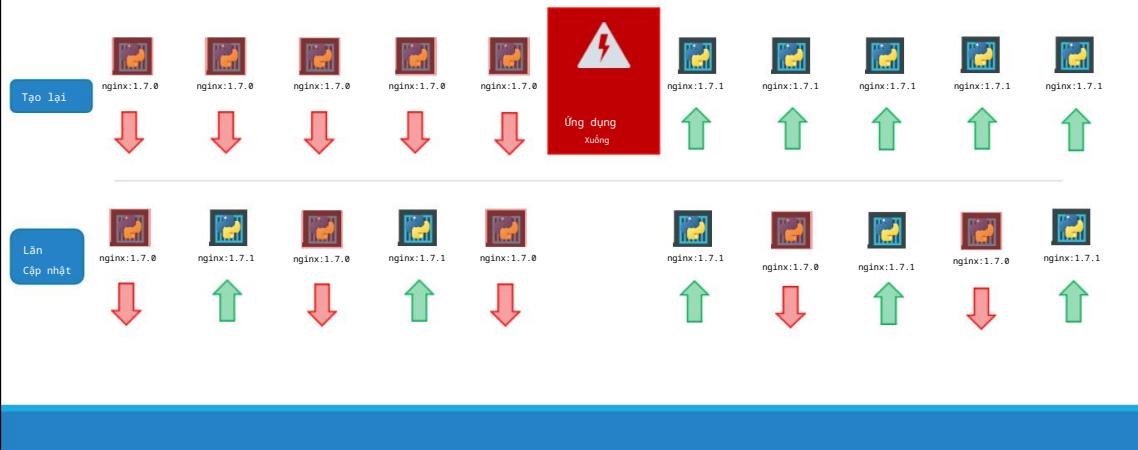
```
> triển khai trạng thái triển khai kubectl/triển khai myapp
Đang chờ quá trình triển khai kết thúc: Cố sẵn 0 trong số 10 bản sao cập nhật...
Đang chờ quá trình triển khai kết thúc: Đã có sẵn 1 trong 10 bản cập nhật...
Đang chờ quá trình triển khai kết thúc: 2 trong số 10 bản cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: 3 trong số 10 bản cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: 4 trong số 10 bản cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: 5 trong số 10 bản cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: 6 trong số 10 bản cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: 7 trong số 10 bản cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: 8 trong số 10 bản sao cập nhật có sẵn...
Đang chờ quá trình triển khai kết thúc: đã có sẵn 9 trong số 10 bản sao cập nhật... triển khai "triển khai myapp" đã triển khai thành công
```

```
> lịch sử triển khai kubectl/triển khai myapp
triển khai "triển khai myapp"
Sửa đổi THAY đổi: NGUYỄN Nhữ
1          <không có>
2          kubectl áp dụng --filename=triển khai-dịnh nghĩa.yml --record=true
```

Bạn có thể xem trạng thái triển khai của mình bằng cách chạy lệnh: trạng thái triển khai kubectl theo sau là tên triển khai.

Để xem các bản sửa đổi và lịch sử triển khai, hãy chạy lệnh kubectl rollout history theo sau là tên triển khai và lệnh này sẽ hiển thị cho bạn các bản sửa đổi.

## Chiến lược triển khai



Có hai loại chiến lược triển khai. Ví dụ: bạn có 5 bản sao của phiên bản ứng dụng web được triển khai. Một cách để nâng cấp chúng lên phiên bản mới hơn là hủy tất cả những thứ này và sau đó tạo các phiên bản ứng dụng mới hơn. Nghĩa là trước tiên, hãy hủy 5 phiên bản đang chạy và sau đó triển khai 5 phiên bản mới của phiên bản ứng dụng mới. Vấn đề với điều này như bạn có thể tưởng tượng là trong khoảng thời gian sau khi các phiên bản cũ ngừng hoạt động và trước khi bất kỳ phiên bản mới nào xuất hiện, ứng dụng sẽ ngừng hoạt động và người dùng không thể truy cập được. Chiến lược này được gọi là chiến lược Tái tạo và may mắn thay, đây KHÔNG phải là chiến lược triển khai mặc định.

Chiến lược thứ hai là chúng ta không tiêu diệt tất cả chúng cùng một lúc. Thay vào đó, chúng tôi gỡ bỏ phiên bản cũ hơn và đưa lên từng phiên bản mới hơn. Bằng cách này, ứng dụng không bao giờ ngừng hoạt động và quá trình nâng cấp diễn ra liền mạch.

Hãy nhớ rằng, nếu bạn không chỉ định chiến lược trong khi tạo triển khai, chiến lược đó sẽ coi đó là Cập nhật luân phiên. Nói cách khác, RollingUpdate là Chiến lược triển khai mặc định.

```

tríen khai-dinh-nghia.yml
MUMSHAD MANAMBETH

apiVersion: loại ứng dụng/v1
tríen khai:
  tên: triển khai myapp
  nhãn:
    ứng dụng: myapp
    Kiểu: giao diện người dùng
    thông số kỹ thuật:
      mẫu: siêu dữ liệu:
        tên: myapp-pod
        nhãn:
          ứng dụng: myapp
          Kiểu: giao diện người dùng
        thông số kỹ thuật:
          hộp ứng dụng:
            - tên: nginx-container
              hình ảnh: nginx:1.7.1
        bản sao: 3
        bộ chọn:
        matchLabels:
          Kiểu: giao diện người dùng

```

Vì vậy, chúng tôi đã nói về việc nâng cấp. Chính xác thì bạn cập nhật quá trình triển khai của mình như thế nào? Khi tôi nói cập nhật, nó có thể là những thứ khác nhau, chẳng hạn như cập nhật phiên bản ứng dụng của bạn bằng cách cập nhật phiên bản của vùng chứa docker được sử dụng, cập nhật nhãn của chúng hoặc cập nhật số lượng bản sao, v.v. Vì chúng tôi đã có tệp định nghĩa triển khai nên chúng tôi rất dễ sửa đổi tập tin này. Sau khi thực hiện các thay đổi cần thiết, chúng tôi chạy lệnh kubectl apply để áp dụng các thay đổi. Quá trình triển khai mới được kích hoạt và bản sửa đổi mới của quá trình triển khai được tạo.

Nhưng có một cách KHÁC để làm điều tương tự. Bạn có thể sử dụng lệnh kubectl set image để cập nhật hình ảnh của ứng dụng. Nhưng hãy nhớ rằng, thực hiện theo cách này sẽ dẫn đến tệp định nghĩa triển khai có cấu hình khác. Vì vậy bạn phải cẩn thận khi sử dụng cùng một tệp định nghĩa để thực hiện các thay đổi trong tương lai.

MUMSHAD MANNABETH

```
.\|> kubectl describe deployment myapp-deployment
Name:           myapp-deployment
Namespace:      default
CreationTimestamp: Sat, 03 Mar 2018 17:01:55 +0000
Labels:         app=app
Annotations:    deployment.kubernetes.io/revision=2
                kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1","kind":"Deployment","metadata":{"name": "myapp-deployment","namespace": "default"}, "spec": {"selector": {"type": "front-end"}, "replicas": 5, "template": {"labels": {"app": "app"}, "spec": {"containers": [{"name": "nginx", "image": "nginx:1.7.1", "ports": [{"port": 80}], "resources": {}, "volumeMounts": [{"name": "empty"}, {"name": "empty"}, {"name": "empty"}], "livenessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 0}, "readinessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 0}, "terminationProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 0}, "restartPolicy": "Always", "securityContext": {}}}}}, "status": {"availableReplicas": 5, "desiredReplicas": 5, "fullyLabeledReplicas": 5, "observedGeneration": 2, "readyReplicas": 5, "replicas": 5, "unavailableReplicas": 0}, "events": [{"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 5"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1m", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-679584ab58 to 0"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "56s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-5dc7d6cc to 5"}]}
```

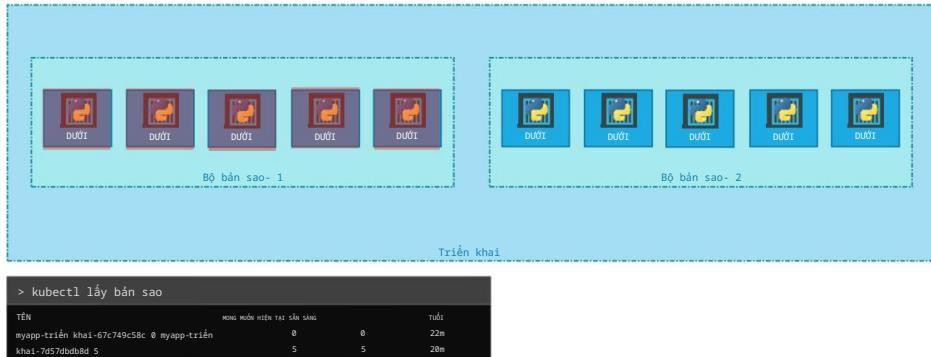
```
.\|> kubectl describe deployment myapp-deployment
Name:           myapp-deployment
Namespace:      default
CreationTimestamp: Sat, 10 Mar 2018 17:16:53 +0000
Labels:         app=app
Annotations:    deployment.kubernetes.io/revision=3
                kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1","kind":"Deployment","metadata":{"name": "myapp-deployment","namespace": "default"}, "spec": {"selector": {"type": "front-end"}, "replicas": 5, "template": {"labels": {"app": "app"}, "spec": {"containers": [{"name": "nginx", "image": "nginx:1.7.1", "ports": [{"port": 80}], "resources": {}, "volumeMounts": [{"name": "empty"}, {"name": "empty"}, {"name": "empty"}], "livenessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 0}, "readinessProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 0}, "terminationProbe": {"httpGet": {"path": "/"}, "initialDelaySeconds": 0}, "restartPolicy": "Always", "securityContext": {}}}}}, "status": {"availableReplicas": 5, "desiredReplicas": 5, "fullyLabeledReplicas": 5, "observedGeneration": 3, "readyReplicas": 5, "replicas": 5, "unavailableReplicas": 0}, "events": [{"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 5"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 2"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 4"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-679584ab58 to 3"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 4"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 3"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 2"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled up replica set myapp-deployment-679584ab58 to 1"}, {"type": "Normal", "reason": "ScalingReplicaSet", "age": "1s", "from": "deployment-controller", "message": "Scaled down replica set myapp-deployment-679584ab58 to 0"}]}
```

Tạo lại

Cập nhật lần

Bạn cũng có thể thấy sự khác biệt giữa chiến lược tạo lại và cập nhật luân phiên khi bạn xem chi tiết các hoạt động triển khai. Chạy lệnh triển khai mô tả kubectl để xem thông tin chi tiết về việc triển khai. Bạn sẽ nhận thấy khi chiến lược Tái tạo được sử dụng, các sự kiện cho biết rằng bộ bản sao cũ được thu nhỏ xuống 0 trước tiên và bộ bản sao mới được thu nhỏ lên 5. Tuy nhiên, khi sử dụng chiến lược RollingUpdate, bộ bản sao cũ được thu nhỏ cùng lúc nhân rộng từng bộ bản sao mới.

## Nâng cấp



Hãy xem cách triển khai thực hiện nâng cấp một cách chi tiết. Khi một triển khai mới được tạo, chẳng hạn như triển khai 5 bản sao, trước tiên, nó sẽ tự động tạo một Bản sao, từ đó tạo ra số lượng POD cần thiết để đáp ứng số lượng bản sao. Khi bạn nâng cấp ứng dụng của mình như chúng ta đã thấy trong trang trình bày trước, đối tượng triển khai kubernetes sẽ tạo một bản sao MỚI bên trong và bắt đầu triển khai các vùng chứa ở đó. Đồng thời gỡ bỏ các POD trong bộ bản sao cũ theo chiến lược RollingUpdate.

Bạn có thể thấy điều này khi cố gắng liệt kê các bản sao bằng lệnh `kubectl get replicasesets`. Ở đây chúng ta thấy bản sao cũ có 0 POD và bản sao mới có 5 POD.

## Khôi phục



Ví dụ: khi bạn nâng cấp ứng dụng của mình, bạn nhận ra có điều gì đó không ổn. Đã xảy ra lỗi với phiên bản mới của bản dựng mà bạn dùng để nâng cấp. Vì vậy, bạn muốn khôi phục bản cập nhật của mình. Việc triển khai Kubernetes cho phép bạn quay lại bản sửa đổi trước đó. Để hoàn tác một thay đổi, hãy chạy lệnh kubectl rollout undo theo sau là tên của quá trình triển khai. Việc triển khai sau đó sẽ phá hủy POD trong bản sao mới và đưa những cái cũ hơn vào bản sao cũ. Và ứng dụng của bạn trở lại định dạng cũ hơn.

Khi so sánh kết quả đầu ra của lệnh kubectl get replicsets, trước và sau khi khôi phục, bạn sẽ có thể nhận thấy sự khác biệt này. Trước khi khôi phục, bản sao đầu tiên có 0 POD và bản sao mới có 5 POD và điều này sẽ bị đảo ngược sau khi quá trình khôi phục kết thúc.

## chạy kubectl

```
> kubectl chạy nginx --image=nginx  
đã tạo triển khai "nginx"
```

Và cuối cùng, hãy quay lại một trong những lệnh chúng ta đã chạy ban đầu khi lần đầu tiên tìm hiểu về POD. Chúng tôi đã sử dụng lệnh chạy kubectl để tạo POD.

Lệnh này thực tế tạo ra một triển khai chứ không chỉ là POD. Đây là lý do tại sao đầu ra của lệnh cho biết Deployment nginx đã được tạo. Đây là một cách khác để tạo triển khai bằng cách chỉ xác định tên hình ảnh và không sử dụng tệp định nghĩa. Một bản sao và nhóm được tạo tự động trong phần phụ trợ. Tuy nhiên, bạn nên sử dụng tệp định nghĩa vì bạn có thể lưu tệp, kiểm tra tệp đó vào kho lưu trữ mã và sửa đổi tệp sau này nếu cần.

## Tóm tắt các lệnh

Tạo nền

```
> kubectl tạo -f triển khai-dịnh nghĩa.yml
```

Lấy

```
> kubectl nhận triển khai
```

Cập nhật

```
> kubectl áp dụng -f triển khai-dịnh nghĩa.yml
```

```
> kubectl set triển khai image/myapp-deployment nginx=nginx:1.9.1
```

Trạng thái

```
> triển khai trạng thái triển khai kubectl/triển khai myapp
```

```
> lịch sử triển khai kubectl/triển khai myapp
```

Khôi phục

```
> triển khai kubectl hoàn tác triển khai/triển khai myapp
```

Để tóm tắt các lệnh thật nhanh, hãy sử dụng lệnh tạo kubectl để tạo triển khai, nhận lệnh triển khai để liệt kê các triển khai, áp dụng và đặt lệnh hình ảnh để cập nhật các hoạt động triển khai, lệnh trạng thái triển khai để xem trạng thái triển khai và lệnh hoàn tác triển khai để khôi phục một hoạt động triển khai.



## Thử nghiệm

Triển khai

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.

## Mạng 101

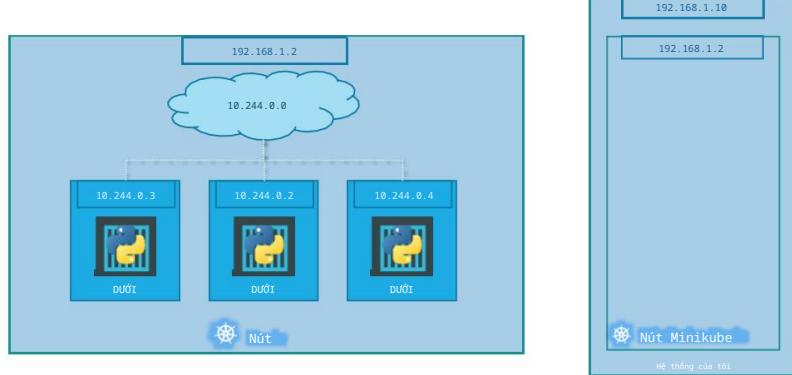


mumshad.mannambeth

Trong bài giảng này, chúng ta sẽ thảo luận về Mạng trong kubernetes.

# Mạng Kubernetes - 101

- Địa chỉ IP được gán cho POD



Chúng ta hãy xem xét những điều cơ bản nhất về kết nối mạng trong Kubernetes. Chúng ta sẽ bắt đầu với một cụm Kubernetes một nút. Nút có địa chỉ IP, giả sử nó là 192.168.1.2 trong trường hợp này. Đây là địa chỉ IP mà chúng tôi sử dụng để truy cập nút Kubernetes, SSH vào nút đó, v.v. Ngoài ra, hãy nhớ rằng nếu bạn đang sử dụng thiết lập Minikube thì tôi đang nói về địa chỉ IP của máy ảo minikube bên trong Hypervisor của bạn. Máy tính xách tay của bạn có thể có IP khác như 192.168.1.10. Vì vậy, điều quan trọng là phải hiểu cách thiết lập máy ảo.

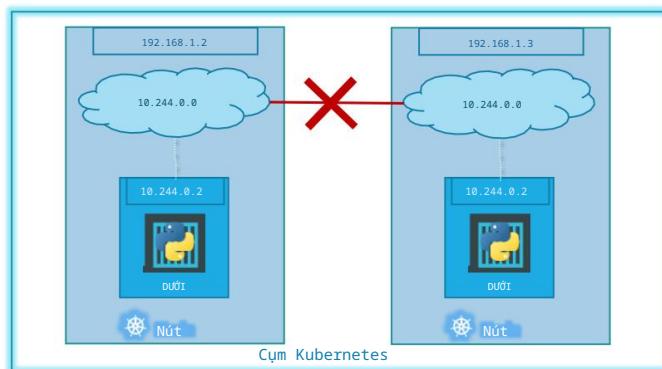
Vì vậy, trên cụm Kubernetes nút đơn, chúng tôi đã tạo một POD đơn. Như bạn đã biết POD lưu trữ một container. Không giống như trong thế giới docker, địa chỉ IP luôn được gán cho Docker CONTAINER,

trong Kubernetes, địa chỉ IP được gán cho POD. Mỗi POD trong Kubernetes có Địa chỉ IP nội bộ riêng. Trong trường hợp này, nó nằm trong dãy 10.244 và IP được gán cho POD là 10.244.0.2. Vậy làm thế nào nó có được địa chỉ IP này? Khi Kubernetes được định cấu hình ban đầu, nó sẽ tạo một mạng riêng nội bộ có địa chỉ 10.244.0.0 và tất cả POD được gán vào mạng đó. Khi bạn triển khai nhiều POD, tất cả chúng đều được gán một IP riêng. Các POD có thể liên lạc với nhau thông qua IP này. Nhưng việc truy cập các POD khác bằng địa chỉ IP nội bộ này CÓ THỂ không phải là một ý tưởng hay vì nó có thể thay đổi khi POD được tạo lại. Chúng ta sẽ thấy những cách TỐT HƠN để thiết lập

liên lạc giữa các POD trong một thời gian. Hiện tại, điều quan trọng là phải hiểu cách hoạt động của mạng nội bộ trong Kubernetes.

## Mạng cụm

- Tắt cả các container/POD có thể giao tiếp với một cái khác không có NAT
- Tắt cả các nút có thể giao tiếp với tất cả container và ngược lại không có NAT



Vì vậy, thật dễ dàng và đơn giản để hiểu khi kết nối mạng trên một nút duy nhất. Nhưng nó hoạt động như thế nào khi bạn có nhiều nút trong một cụm? Trong trường hợp này, chúng ta có hai nút chạy kubernetes và chúng được gán địa chỉ IP 192.168.1.2 và 192.168.1.3. Lưu ý rằng chúng chưa thuộc cùng một cụm.

Mỗi người trong số họ có một POD duy nhất được triển khai. Như đã thảo luận ở slide trước, những nhóm được gắn vào mạng nội bộ và chúng được chỉ định địa chỉ IP riêng. TUY NHIÊN, nếu bạn nhìn vào các địa chỉ mạng, bạn có thể thấy chúng giống nhau. Hai mạng có địa chỉ 10.244.0.0 và POD được triển khai cũng có cùng địa chỉ.

Điều này SẼ KHÔNG hoạt động tốt khi các nút là một phần của cùng một cụm. Các POD có cùng địa chỉ IP được gán cho chúng và điều đó sẽ dẫn đến xung đột IP trong mạng. Bây giờ đó là MỘT vấn đề. Khi cụm kubernetes được THIẾT LẬP, kubernetes KHÔNG tự động thiết lập bất kỳ loại mạng nào để xử lý các sự cố này. Trên thực tế, kubernetes mong muốn Hoa Kỳ thiết lập mạng để đáp ứng các yêu cầu cơ bản nhất định. Một số trong số này là tắt cả các thùng chứa hoặc POD trong cụm kubernetes PHẢI có khả năng giao tiếp với nhau mà không cần phải định cấu hình NAT. Tất cả các nút phải có khả năng giao tiếp với các vùng chứa và tất cả các vùng chứa phải có khả năng giao tiếp với các nút trong cụm. Kubernetes

kỳ vọng Hoa Kỳ sẽ thiết lập một giải pháp mạng đáp ứng các tiêu chí này.



May mắn thay, chúng ta không phải tự mình thiết lập TẤT CẢ vì có sẵn nhiều giải pháp dựng sẵn. Một số trong số đó là mạng Cisco ACI, Cilium, Big Cloud Fabric, Flannel, VMware NSX-t và Calico. Tùy thuộc vào nền tảng bạn đang triển khai cụm Kubernetes, bạn có thể sử dụng bất kỳ giải pháp nào trong số này. Ví dụ: nếu bạn đang thiết lập cụm Kubernetes từ đầu trên hệ thống của riêng mình, bạn có thể sử dụng bất kỳ giải pháp nào trong số này như Calico, Flannel, v.v. Nếu bạn đang triển khai trên môi trường VMware thì NSX-T có thể là một lựa chọn tốt. Nếu bạn nhìn vào phòng thí nghiệm play-with-k8s, họ sử dụng WeaveNet. Trong các bản demo của khóa học, chúng tôi đã sử dụng Calico.

Tùy thuộc vào môi trường của bạn và sau khi đánh giá Ưu và nhược điểm của từng phương pháp những điều này, bạn có thể chọn giải pháp mạng phù hợp.

# Thiết lập mạng cụm

## (3/4) Installing a pod network

You **MUST** install a pod **network** add-on so that your pods can communicate with each other.

The **network** must be deployed before any applications. Also, **kube-dns**, an internal helper service, will not start up before a **network** is installed. **kubeadm** only supports Container Network Interface (CNI) based **networks** (and does not support kubenet).

Several projects provide Kubernetes pod **networks** using CNI, some of which also support **Network Policy**. See the [pod network](#) for a complete list of available **network** add-ons. IPv6 support was added in [v1.7.0](#). [Calico](#) and [Weave](#) are the only supported IPv6 **network** plugins in 1.9.

**Note:** **kubeadm** sets up a more secure cluster by default and enforces use of [RBAC](#). Please make sure that the **network** manifest of choice supports RBAC.

You can install a pod **network** add-on with the following command:

```
kubectl apply -f <add-on.yaml>
```

**NOTE:** You can install **only one** pod **network** per cluster.

Choose one... [Calico](#) [Canal](#) [Flannel](#) [Kube-router](#) [Romana](#) [Weave Net](#)

Refer to the Calico documentation for a [kubeadm quickstart](#), a [kubeadm installation guide](#), and other resources.

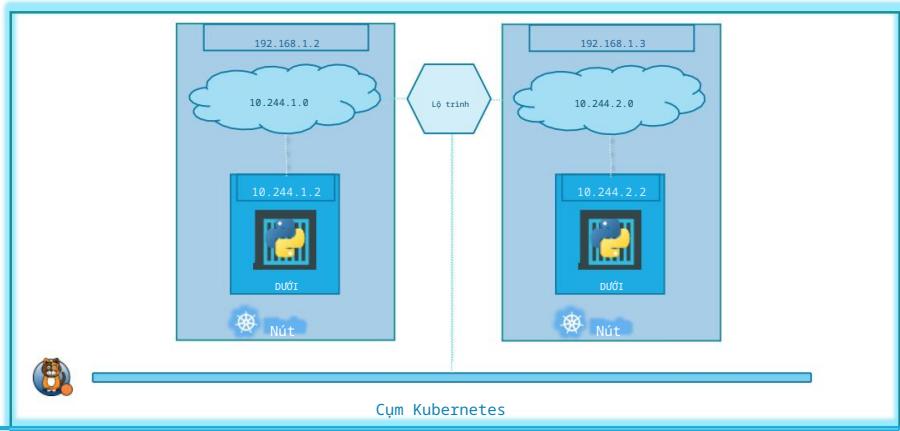
**Note:**

- In order for Network Policy to work correctly, you need to pass `--pod-network-cidr=192.168.0.0/16` to `kubeadm init`.
- Calico works on [amd64](#) only.

```
kubectl apply -f https://docs.projectcalico.org/v3.8/getting-started/kubernetes/installation/hosted/kubeadm/1.7/calico.yaml
```

Nếu bạn quay lại và xem bản demo ban đầu chúng tôi thiết lập cụm kubernetes, chúng tôi thiết lập mạng dựa trên Calico. Chúng tôi chỉ mất một vài bộ lệnh để thiết lập nó. Vì vậy, nó khá dễ dàng.

## Mạng cụm



Vì vậy, hãy quay lại cụm của chúng tôi, với thiết lập mạng Calico, giờ đây nó quản lý các mạng và IPs trong các nút của tôi và chỉ định một địa chỉ mạng khác nhau cho mỗi mạng trong các nút. Điều này tạo ra một mạng ảo gồm tất cả các POD và nút vì tất cả chúng đều được gán một Địa chỉ IP duy nhất. Và bằng cách sử dụng các kỹ thuật định tuyến đơn giản, mạng cụm cho phép liên lạc giữa các POD hoặc Nút khác nhau để đáp ứng các yêu cầu kết nối mạng của Kubernetes. Do đó, tất cả các POD hiện có thể liên lạc với nhau bằng địa chỉ IP được chỉ định.

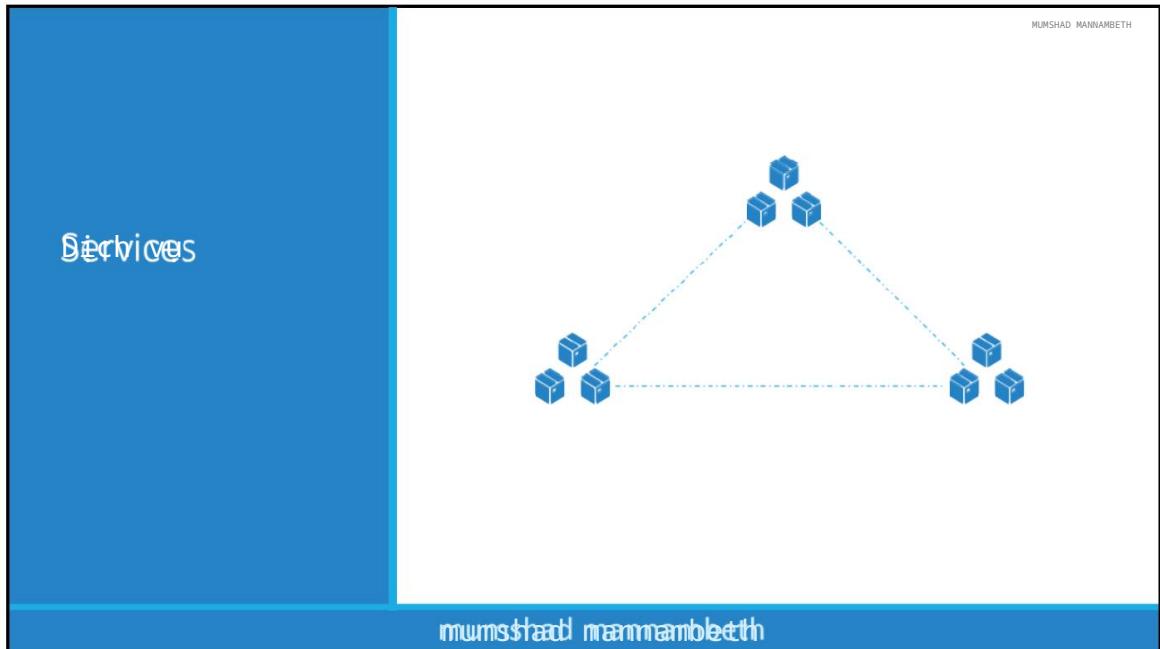


## Thử nghiệm

---

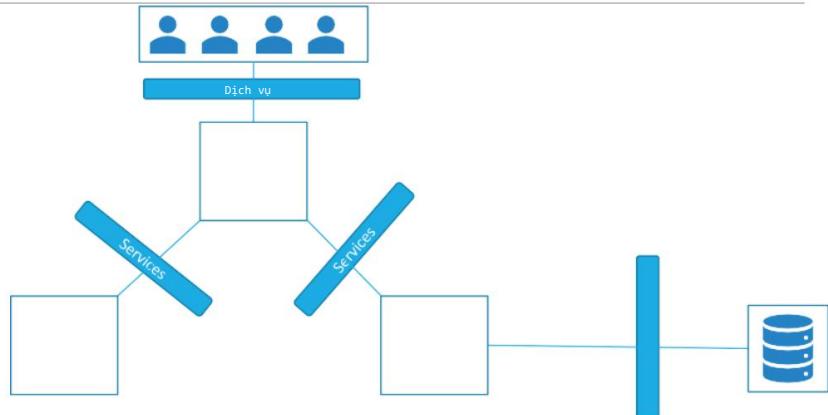
Mạng

Đó là chỗ cho bài giảng này. Bây giờ chúng ta sẽ chuyển sang phần Demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.



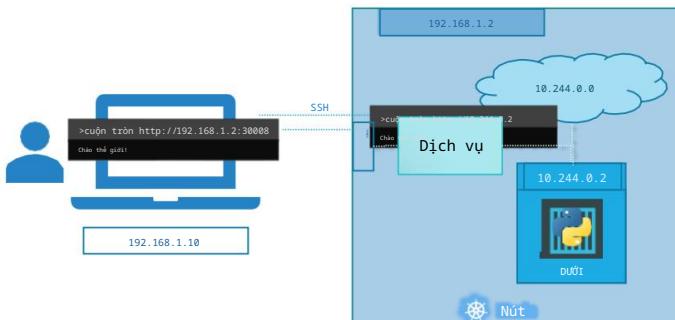
Trong bài giảng này, chúng ta sẽ thảo luận về Dịch vụ Kubernetes.

## Dịch vụ



Dịch vụ Kubernetes cho phép giao tiếp giữa các thành phần khác nhau trong và ngoài ứng dụng. Dịch vụ Kubernetes giúp chúng tôi kết nối các ứng dụng với nhau với các ứng dụng hoặc người dùng khác. Ví dụ: Ứng dụng của chúng tôi có các nhóm POD chạy nhiều phần khác nhau, chẳng hạn như một nhóm để phân phát tải giao diện người dùng cho người dùng, một nhóm khác chạy các quy trình phụ trợ và nhóm thứ ba kết nối với nguồn dữ liệu bên ngoài. Chính Dịch vụ cho phép kết nối giữa các nhóm POD này. Các dịch vụ cho phép cung cấp ứng dụng giao diện người dùng cho người dùng, nó giúp liên lạc giữa các POD mặt sau và mặt trước, đồng thời giúp thiết lập kết nối với nguồn dữ liệu bên ngoài. Do đó, các dịch vụ cho phép kết nối lỏng lẻo giữa các vi dịch vụ trong ứng dụng của chúng ta.

## Dịch vụ



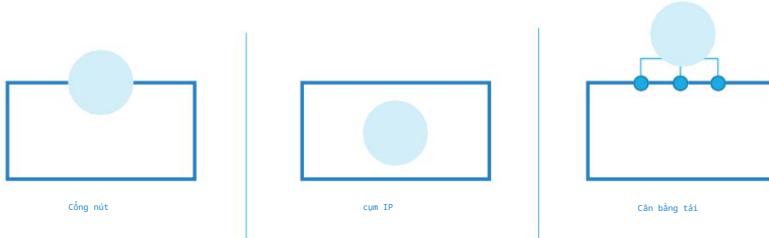
Chúng ta hãy xem xét một trường hợp sử dụng Dịch vụ. Cho đến nay chúng ta đã nói về cách các POD giao tiếp với nhau thông qua mạng nội bộ. Chúng ta hãy xem xét một số khía cạnh khác của mạng trong bài giảng này. Hãy bắt đầu với giao tiếp bên ngoài. Vì vậy chúng tôi đã triển khai POD của chúng tôi có ứng dụng web chạy trên đó. CHÚNG TÔI với tư cách là người dùng bên ngoài truy cập trang web bằng cách nào? Trước hết, hãy nhìn vào thiết lập hiện có. Nút Kubernetes có địa chỉ IP là 192.168.1.2. Máy tính xách tay của tôi cũng nằm trên cùng một mạng nên nó có địa chỉ IP 192.168.1.10. Mạng POD nội bộ nằm trong phạm vi 10.244.0.0 và POD có IP 10.244.0.2. Rõ ràng, tôi không thể ping hoặc truy cập POD tại địa chỉ 10.244.0.2 vì nó nằm trong một mạng riêng. Vậy các tùy chọn để xem trang web là gì?

Đầu tiên, nếu chúng ta SSH vào nút kubernetes tại 192.168.1.2, từ nút đó, chúng ta sẽ có thể truy cập trang web của POD bằng cách thực hiện cuộn tròn hoặc nếu nút có GUI, chúng ta có thể kích hoạt trình duyệt và xem trang web trong trình duyệt có địa chỉ http://10.244.0.2. Nhưng đây là từ bên trong Kubernetes Node và đó không phải là điều tôi thực sự muốn. Tôi muốn có thể truy cập máy chủ web từ máy tính xách tay của mình mà không cần phải SSH vào nút và chỉ cần truy cập IP của nút kubernetes.

Vì vậy, chúng tôi cần một cái gì đó ở giữa để giúp chúng tôi ánh xạ các yêu cầu tới nút từ máy tính xách tay của chúng tôi thông qua nút tới POD chạy vùng chứa web.

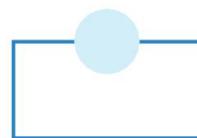
Đó là lúc dịch vụ kubernetes phát huy tác dụng. Dịch vụ kubernetes là một đối tượng giống như POD, Replicaset hoặc Deployments mà chúng ta đã làm việc trước đây. Một trong những trường hợp sử dụng của nó là lắng nghe một cổng trên Nút và chuyển tiếp các yêu cầu trên cổng đó tới một cổng trên POD chạy ứng dụng web. Loại dịch vụ này được gọi là dịch vụ NodePort vì dịch vụ này lắng nghe một cổng trên Node và chuyển tiếp yêu cầu tới POD. Có những loại dịch vụ khác hiện có mà chúng ta sẽ thảo luận.

## Loại dịch vụ



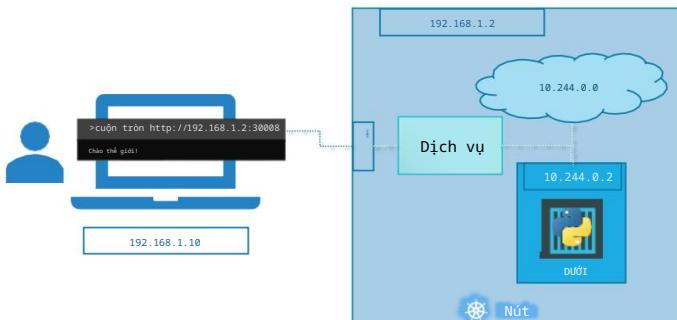
Vấn đề đầu tiên là những gì chúng ta đã thảo luận - NodePort là dịch vụ giúp POD nội bộ có thể truy cập được trên một Cổng trên Node. Thứ hai là ClusterIP - và trong này trường hợp dịch vụ tạo một IP ảo bên trong cụm để cho phép liên lạc giữa các dịch vụ khác nhau, chẳng hạn như một tập hợp các máy chủ ngoại vi với một tập hợp các máy chủ phụ trợ. Loại thứ ba là LoadBalancer, nó cung cấp bộ cân bằng tải cho dịch vụ của chúng tôi trong các nhà cung cấp đám mây được hỗ trợ. Một ví dụ điển hình về điều đó là phân phối tải trên các máy chủ web khác nhau. Vậy giờ chúng ta sẽ xem xét từng thứ này chi tiết hơn một chút cùng với một số Bản demo.

Cổng nút



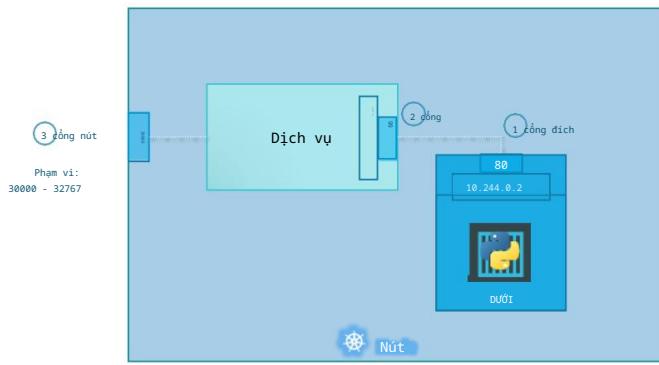
Trong bài giảng này, chúng ta sẽ thảo luận về Dịch vụ NodePort Kubernetes.

## Dịch vụ - NodePort



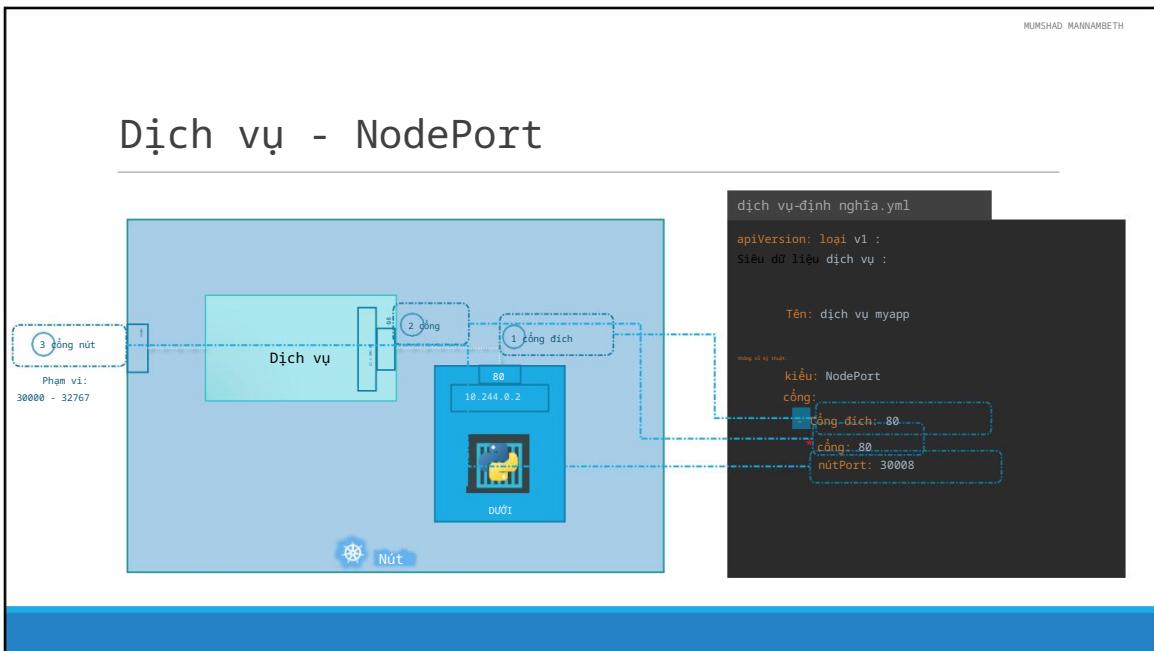
Quay lại với NodePort, trong một vài trang trình bày phía trước chúng ta đã thảo luận về quyền truy cập từ bên ngoài vào ứng dụng. Chúng tôi đã nói rằng Dịch vụ có thể giúp chúng tôi bằng cách ánh xạ một cổng trên Nút tới một cổng trên POD.

## Dịch vụ - NodePort



Chúng ta hãy xem kỹ hơn về Dịch vụ. Nếu bạn nhìn vào nó, có 3 cổng liên quan. Cổng trên POD mà máy chủ web thực tế đang chạy là cổng 80. Và nó được gọi là Cổng đích vì đó là dịch vụ chuyển tiếp yêu cầu tới. Cổng thứ hai là cổng trên chính dịch vụ đó. Nó được gọi đơn giản là cổng.

Hãy nhớ rằng, những điều khoản này là từ quan điểm của dịch vụ. Trên thực tế, dịch vụ này giống như một máy chủ ảo bên trong nút. Bên trong cụm nó có địa chỉ IP riêng. Và địa chỉ IP đó được gọi là Cluster-IP của dịch vụ. Và cuối cùng chúng ta có cổng trên chính Node mà chúng ta sử dụng để truy cập máy chủ web từ bên ngoài. Và nó được gọi là NodePort. Như bạn có thể thấy, nó là 30008. Đó là bởi vì NodePort chỉ có thể nằm trong phạm vi hợp lệ từ 30000 đến 32767.



Bây giờ chúng ta hãy xem cách tạo ra dịch vụ. Giống như cách chúng tôi tạo Triển khai, Bản sao hoặc Pod, chúng tôi sẽ sử dụng tệp định nghĩa để tạo dịch vụ.

Cấu trúc cấp cao của tập tin vẫn giữ nguyên. Như trước đây, chúng tôi có các phần apiVersion, loại, siêu dữ liệu và thông số kỹ thuật. apiVersion sẽ là v1. Loại này tất nhiên là dịch vụ. Siêu dữ liệu sẽ có tên và đó sẽ là tên của dịch vụ. Nó có thể có nhãn, nhưng hiện tại chúng ta chưa cần điều đó. Tiếp theo chúng ta có thông số kỹ thuật. và như mọi khi, đây là phần quan trọng nhất của tệp vì đây là lúc chúng ta sẽ xác định các dịch vụ thực tế và đây là phần của tệp định nghĩa khác nhau giữa các đối tượng khác nhau. Trong phần thông số kỹ thuật của một dịch vụ, chúng tôi có loại và cổng. Loại đề cập đến loại dịch vụ chúng tôi đang tạo. Như đã thảo luận trước đó, nó có thể là ClusterIP, NodePort hoặc LoadBalancer. Trong trường hợp này vì chúng ta đang tạo NodePort nên chúng ta sẽ đặt nó là NodePort. Phần tiếp theo của thông số kỹ thuật là cổng. Đây là nơi chúng tôi nhập thông tin liên quan đến những gì chúng tôi đã thảo luận ở phía bên trái của màn hình này. Loại cổng đầu tiên là targetPort, chúng tôi sẽ đặt thành 80. Loại tiếp theo chỉ đơn giản là cổng, là cổng trên đối tượng dịch vụ và chúng tôi cũng sẽ đặt cổng đó thành 80. Thứ ba là NodePort mà chúng tôi sẽ đặt thành 30008 hoặc bất kỳ số nào trong phạm vi hợp lệ. Hãy nhớ rằng trong số này, trường hợp bắt buộc là **không có port** cấp Cổng đích thì cổng này được coi là giống với cổng và nếu bạn không cung cấp cổng cổng, cổng còn trống trong phạm vi hợp lệ từ 30000 đến 32767 sẽ tự động được phân bổ. Cũng lưu ý rằng port là một mảng.

Vì vậy hãy lưu ý dấu gạch ngang bên dưới phần cổng cho biết phần tử đầu tiên trong mảng. Bạn có thể có nhiều ánh xạ cổng như vậy trong một dịch vụ.

Vì vậy, chúng tôi có tất cả thông tin, nhưng thực sự còn thiếu một cái gì đó. Không có gì ở đây trong tệp định nghĩa kết nối dịch vụ với POD. Chúng tôi chỉ chỉ định Cổng đích nhưng chưa đề cập đến Cổng đích mà POD trên đó. Có thể có hàng trăm POD khác có dịch vụ web chạy trên cổng 80. Vậy chúng ta làm điều đó như thế nào?

Như chúng ta đã làm với các bản sao trước đây và một kỹ thuật mà bạn sẽ thấy rất thường xuyên trong kubernetes, chúng ta sẽ sử dụng nhãn và bộ chọn để liên kết chúng với nhau. Chúng tôi biết rằng POD đã được tạo bằng nhãn. Chúng ta cần đưa nhãn đó vào tệp định nghĩa dịch vụ này.

## Dịch vụ - NodePort

The screenshot shows a terminal window with two panes. The left pane displays the YAML configuration for a service named 'myapp' with a NodePort type. The right pane shows the output of 'kubectl get services' and the resulting deployment with a NodePort of 30008.

```

dịch vụ-định nghĩa.yml
apiVersion: v1
loại: Dịch vụ
metadata:
  tên: dịch vụ myapp
nhóm số kỹ thuật:
  kiểu: NodePort
cổng:
  - Cổng dịch: 80
    cổng: 80
    nútPort: 30008
bộ chọn:

```

```

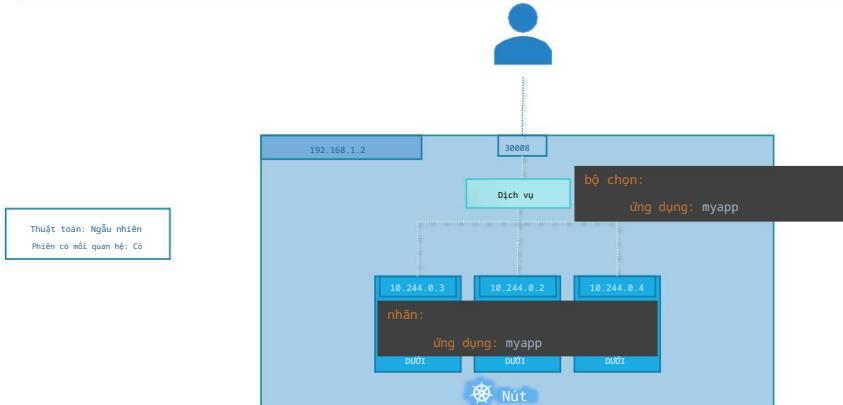
pod-định nghĩa.yml
> kubectl acprieVateers-ifosne:rvvilce-def định nghĩa.yml
dịch vụ "myapp-llinerdvi:ce" được xác nhận
> kubectl nhàn dịch vụ
TÊN          NAME     PORT(S)        ST   IP BÊN NGOÀI      CỔNG      TUỔI
kubernetes   kubernetes   443/TCP       10.96.0.1   <không c>   443/TCP   14 ngày
myapp-service   lWebbedPloar: 10.106.127.123   <không c>   80:30008/TCP 5m
  ứng dụng: myapp
  > cuộn tròn http://192.168.1.2:30008
<html>
<head>
  <title>Welcome to nginx!</title>
<style>
  - tên: nginx-container
    body {
      width: 35em;
      margin: 0 auto;
      font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>

```

Vì vậy, chúng ta có một thuộc tính mới trong phần thông số kỹ thuật và đó là bộ chọn. Trong bộ chọn, cung cấp danh sách các nhãn để xác định POD. Để biết điều này, hãy tham khảo tệp định nghĩa nhóm được sử dụng để tạo POD. Kéo nhãn từ tệp định nghĩa nhóm và đặt nó dưới phần bộ chọn. Điều này liên kết dịch vụ với nhóm. Sau khi hoàn tất, hãy tạo dịch vụ bằng lệnh `kubectl create` và nhập tệp định nghĩa dịch vụ và bạn đã tạo dịch vụ ở đó.

Để xem dịch vụ đã tạo, hãy chạy lệnh `kubectl get services` để liệt kê các dịch vụ, cluster-ip của chúng và các cổng được ánh xạ. Loại là NodePort như chúng tôi đã tạo và cổng trên nút được gán tự động là 32432. Bây giờ chúng tôi có thể sử dụng cổng này để truy cập dịch vụ web bằng cách sử dụng `Curl` hoặc trình duyệt web.

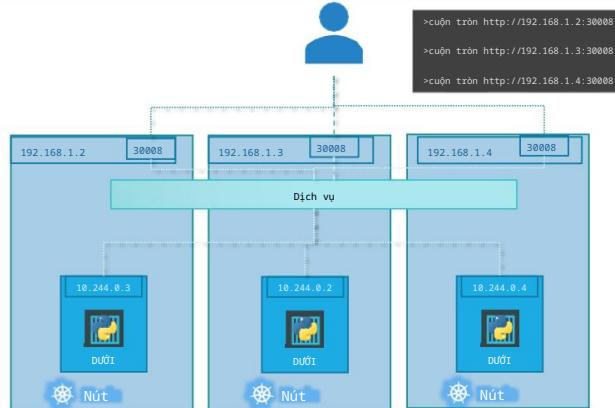
## Dịch vụ - NodePort



Cho đến nay chúng ta đã nói về một dịch vụ được ánh xạ tới một POD duy nhất. Nhưng không phải lúc nào cũng như vậy, bạn sẽ làm gì khi có nhiều POD? Trong môi trường sản xuất, bạn có nhiều phiên bản ứng dụng web của mình đang chạy nhằm mục đích cân bằng tải và có tính sẵn sàng cao.

Trong trường hợp này, chúng tôi có nhiều POD tương tự đang chạy ứng dụng web của mình. Tất cả chúng đều có cùng nhãn với một ứng dụng chính được đặt thành giá trị myapp. Nhãn tương tự được sử dụng làm bộ chọn trong quá trình tạo dịch vụ. Vì vậy, khi dịch vụ được tạo, nó sẽ tìm các POD phù hợp với nhãn và tìm thấy 3 trong số đó. Sau đó, dịch vụ sẽ tự động chọn tất cả 3 POD làm điểm cuối để chuyển tiếp các yêu cầu bên ngoài đến từ người dùng. Bạn không cần phải thực hiện bất kỳ cấu hình bổ sung nào để thực hiện điều này. Và nếu bạn đang thắc mắc nó sử dụng thuật toán nào để cân bằng tải thì nó sử dụng thuật toán ngẫu nhiên. Do đó, dịch vụ hoạt động như một bộ cân bằng tải tích hợp để phân phối tải trên các POD khác nhau.

## Dịch vụ - NodePort



Và cuối cùng, hãy xem điều gì sẽ xảy ra khi POD được phân phối trên nhiều nút. Trong trường hợp này, chúng tôi có ứng dụng web trên POD trên các nút riêng biệt mà không cần phải thực cụm. Khi chúng tôi tạo một dịch vụ , hiện BẤT KỲ loại cấu hình bổ sung nào, kubernetes tạo ra một dịch vụ trải rộng trên tất cả các nút trong cụm và ánh xạ cổng đích tới CÙNG NodePort trên tất cả các nút trong cụm. Bằng cách này, bạn có thể truy cập ứng dụng của mình bằng IP của bất kỳ nút nào trong cụm và sử dụng cùng số cổng trong trường hợp này là 30008.

Tóm lại - trong BẤT KỲ trường hợp nào, đó là một nhóm trong một nút, nhiều nhóm trên một nút, nhiều nhóm trên nhiều nút, dịch vụ được tạo giống hệt nhau mà bạn không cần phải thực hiện bất kỳ bước bổ sung nào trong quá trình tạo dịch vụ.

Khi POD được xóa hoặc thêm, dịch vụ sẽ tự động được cập nhật khiến dịch vụ này có tính linh hoạt và thích ứng cao. Sau khi tạo, thông thường bạn sẽ không phải thực hiện bất kỳ thay đổi cấu hình bổ sung nào.

MUMSHAD MANNAMBETH

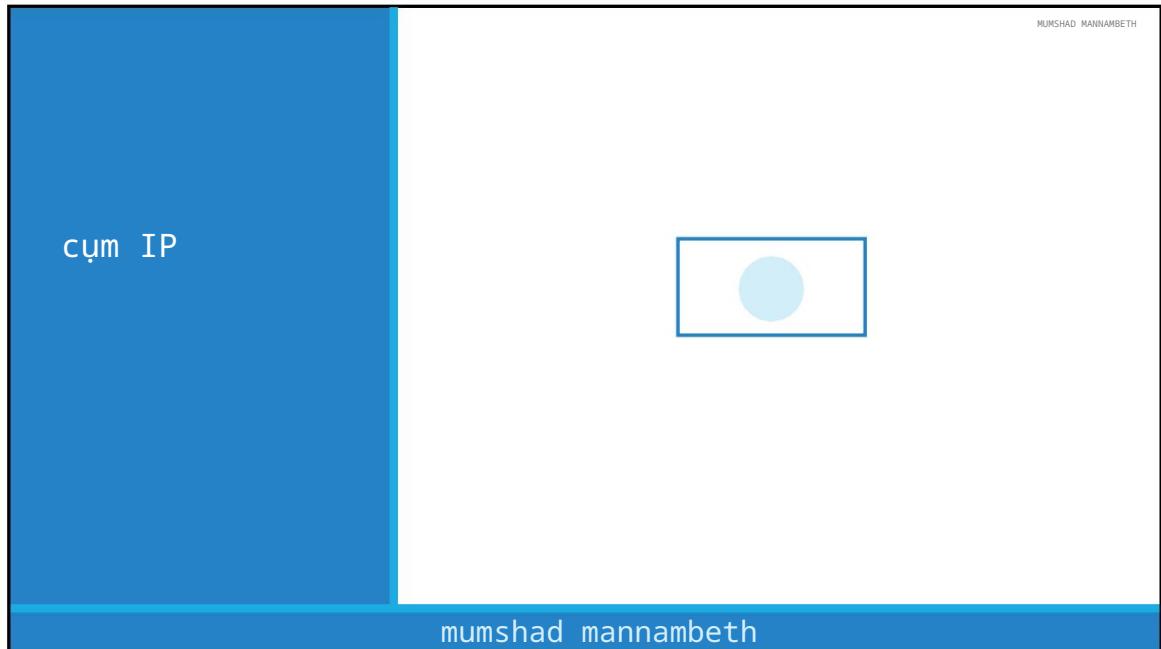


## Thử nghiệm

---

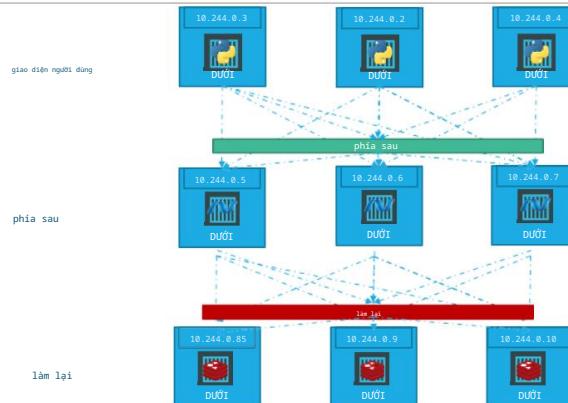
Dịch vụ - NodePort

Bài giảng này thê là xong, hãy xem phần demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.



Trong bài giảng này chúng ta sẽ thảo luận về Kubernetes Service - ClusterIP.

## cụm IP



Một ứng dụng web full stack thường có các loại POD khác nhau lưu trữ các phần khác nhau của ứng dụng. Bạn có thể có một số POD chạy máy chủ web ngoại vi, một tập hợp POD khác chạy máy chủ phụ trợ, một tập hợp POD chạy kho lưu trữ giá trị khóa như Redis, một tập hợp POD khác chạy cơ sở dữ liệu liên tục như MySQL, v.v. -end server cần kết nối với backend-workers và backend-workers cần kết nối với cơ sở dữ liệu cũng như các dịch vụ redis. Vậy đâu là cách phù hợp để thiết lập kết nối giữa các POD này?

Tất cả các POD đều có địa chỉ IP được gán cho chúng như chúng ta có thể thấy trên màn hình. Nhưng những IP này như chúng ta biết không phải là tĩnh, những POD này có thể ngừng hoạt động bất cứ lúc nào và các POD mới luôn được tạo - và vì vậy bạn KHÔNG THỂ dựa vào những địa chỉ IP này để liên lạc nội bộ trong ứng dụng. Ngoài ra, điều gì sẽ xảy ra nếu POD giao diện người dùng đầu tiên tại 10.244.0.3 cần kết nối với dịch vụ phụ trợ? Nó sẽ thuộc về ai trong số 3 và ai là người đưa ra quyết định đó?

Dịch vụ Kubernetes có thể giúp chúng ta nhóm các POD này lại với nhau và cung cấp một giao diện duy nhất để truy cập các POD trong một nhóm. Ví dụ: một dịch vụ được tạo cho các POD phụ trợ sẽ giúp nhóm tất cả các POD phụ trợ lại với nhau và cung cấp một giao diện duy nhất để các POD khác truy cập dịch vụ này. Các yêu cầu được chuyển tiếp đến một trong

các POD theo dịch vụ một cách ngẫu nhiên. Tương tự, tạo các dịch vụ bổ sung cho Redis và cho phép các POD phụ trợ truy cập vào hệ thống redis thông qua dịch vụ này. Điều này cho phép chúng tôi triển khai ứng dụng dựa trên vi dịch vụ trên cụm kubernetes một cách dễ dàng và hiệu quả. Giờ đây, mỗi lớp có thể mở rộng quy mô hoặc di chuyển theo yêu cầu mà không ảnh hưởng đến giao tiếp giữa các dịch vụ khác nhau. Mỗi dịch vụ nhận được một IP và tên được gán cho nó trong cụm và đó là tên sẽ được các POD khác sử dụng để truy cập dịch vụ. Loại dịch vụ này được gọi là ClusterIP.

```

dịch vụ-định nghĩa.yml
apiVersion: v1 loại:
Dịch vụ
metadata:
    Tên: hậu phương
    thông
        số kỹ thuật: loại: ClusterIP
        cổng:
            - Cổng dịch: 80
            cổng: 80
    bộ chọn:

pod-định nghĩa.yml
> kubectl acprieVateers-ifosne:rvvilce-def định nghĩa.yml
dịch vụ "backk-einnd" dc:rePatoedd

> kubectl n g e t t a s d e a r t v a i c: es
TÊN      kubernetes      naBmCjPp myapp. b yên      ERp - o IPd      (Các) CỔNG IP NGOÀI      TUỔI
phía sau      laCbusetlersIP: 10.106.127.123 <none>      443/TCP      14 ngày
                                         80/TCP      2m

        ứng dụng: myapp
        Kiểu: back-end
        thông số kỹ thuật:
        hộp đựng:
            - tên: nginx-container
            hình ảnh: nginx

```

Để tạo một dịch vụ như vậy, như mọi khi, hãy sử dụng tệp định nghĩa. Trong tệp định nghĩa , dịch vụ, trước tiên hãy sử dụng mẫu mặc định có apiVersion, loại, siêu dữ liệu và thông số kỹ thuật. , loại là Dịch vụ và chúng tôi sẽ đặt tên cho dịch vụ của mình - chúng tôi sẽ gọi apiVersion là v1 ở phần back-end. Trong phần Thông số kỹ thuật, chúng tôi có loại và cổng. Loại là ClusterIP. Trên thực tế, ClusterIP là loại mặc định nên ngay cả khi bạn không chỉ định nó, nó sẽ tự động coi đó là ClusterIP. Trong cổng, chúng tôi có cổng đích và cổng. Cổng mục tiêu là cổng mà phần phụ trợ được hiển thị, trong trường hợp này là 80. Và cổng đó là dịch vụ được hiển thị. Đó cũng là 80. Để liên kết dịch vụ với một nhóm POD, chúng tôi sử dụng bộ chọn.

Chúng tôi sẽ tham khảo tệp định nghĩa nhóm và sao chép các nhãn từ đó và di chuyển nó dưới bộ chọn. Và đó nên là nó. Bây giờ chúng ta có thể tạo dịch vụ bằng lệnh kubectl create và sau đó kiểm tra trạng thái của nó bằng lệnh kubectl get services.

Các POD khác có thể truy cập dịch vụ bằng cách sử dụng ClusterIP hoặc tên dịch vụ.

MUMSHAD MANNAMBETH



## Thử nghiệm

---

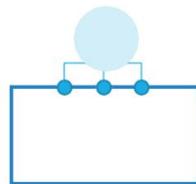
Dịch vụ - NodePort

Bài giảng này thê là xong, hãy xem phần demo và tôi sẽ gặp bạn trong bài giảng tiếp theo.

## Người giới thiệu

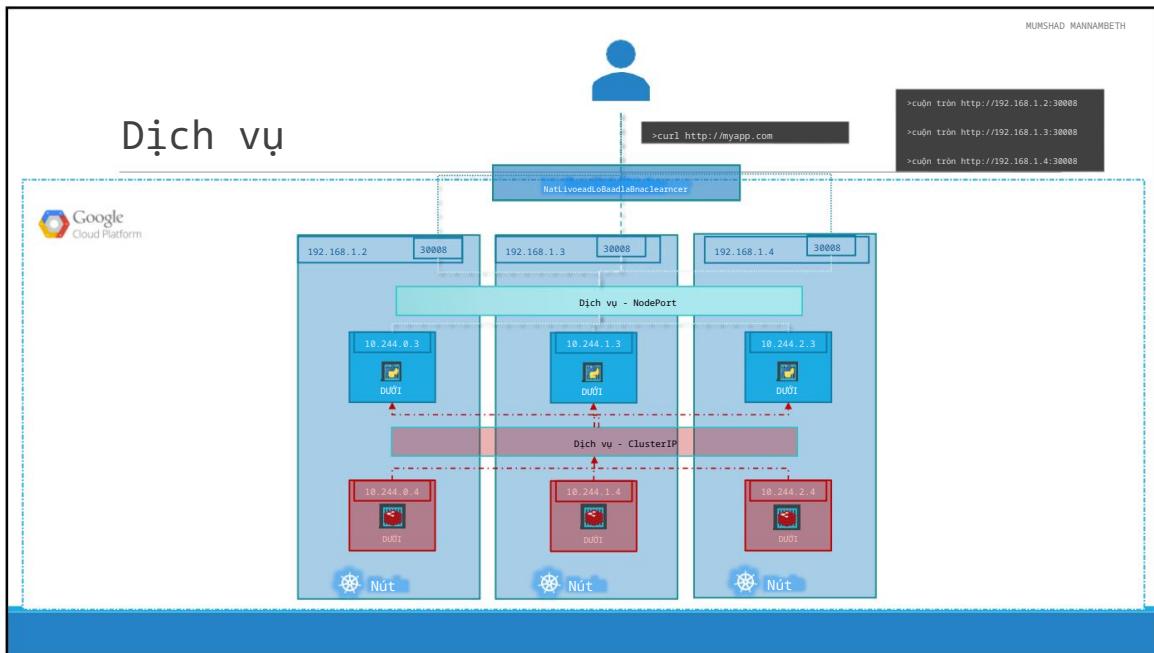
<https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/>

## Dịch vụ - Cân bằng tải



mumshad mannambeth

Trong bài giảng này, chúng ta sẽ thảo luận về loại Dịch vụ Kubernetes thứ ba - LoadBalancer.



Chúng tôi sẽ nhanh chóng tóm tắt lại những gì chúng tôi đã tìm hiểu về hai loại dịch vụ để có thể chuyển sang loại LoadBalancer. Chúng tôi có cụm 3 nút với Ips 192.168.1.2,3 và 4.

Ứng dụng của chúng tôi là hai tầng, có dịch vụ cơ sở dữ liệu và dịch vụ web giao diện người dùng để người dùng truy cập ứng dụng. Loại dịch vụ mặc định -

được gọi là ClusterIP - cung cấp một dịch vụ, chẳng hạn như redis hoặc dịch vụ cơ sở dữ liệu trong nội bộ cụm kubernetes để các ứng dụng khác sử dụng.

Cấp tiếp theo trong ứng dụng của tôi là giao diện người dùng web dựa trên python. Ứng dụng này kết nối với phần phụ trợ bằng Dịch vụ được tạo cho dịch vụ redis. Để hiển thị ứng dụng cho người dùng cuối, chúng tôi tạo một dịch vụ khác thuộc loại NodePort.

Việc tạo một dịch vụ thuộc loại NodePort sẽ hiển thị ứng dụng trên cổng cao cấp của Nút và người dùng có thể truy cập ứng dụng tại bất kỳ IP nào trong các nút của tôi bằng cổng 30008.

Bây giờ, bạn cung cấp cho người dùng cuối IP gì để truy cập ứng dụng của mình? Bạn không thể cho họ cả ba và để họ chọn một trong số họ. Điều người dùng thực sự muốn là một URL duy nhất để truy cập ứng dụng. Để làm được điều này, bạn sẽ được yêu cầu thiết lập một VM Load Balancer riêng biệt trong môi trường của mình. Trong trường hợp này, tôi triển khai một máy ảo mới cho mục đích cân bằng tải và định cấu hình nó để chuyển tiếp các yêu cầu đến bất kỳ máy chủ nào trong số đó.

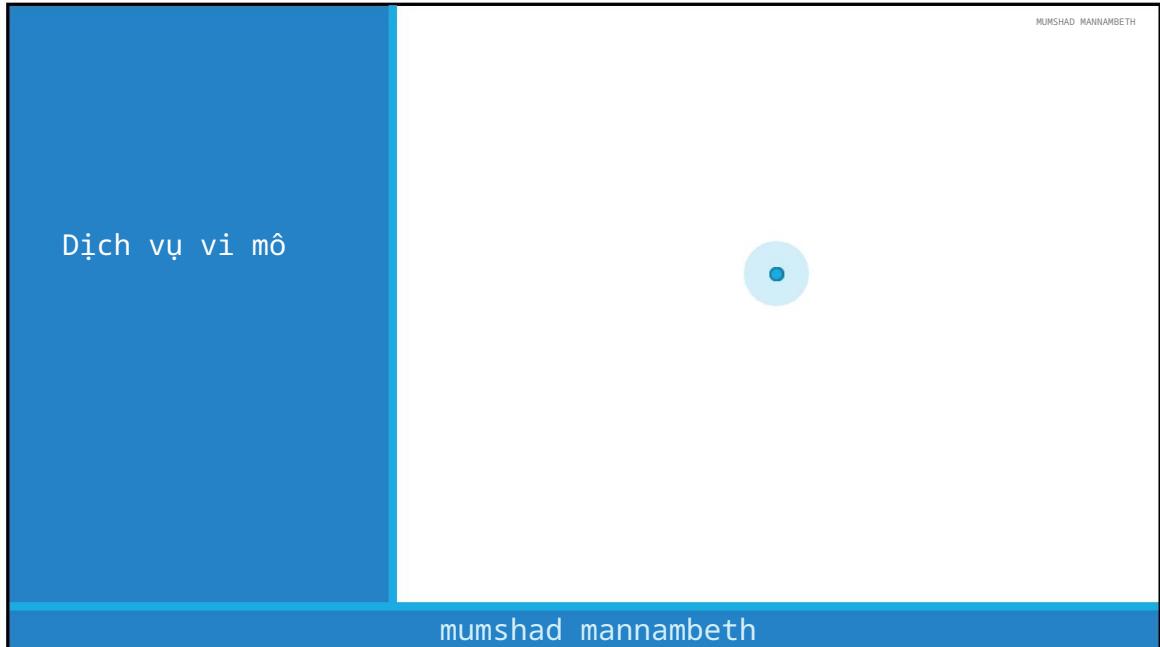
Ips của các nút Kubernetes. Sau đó, tôi sẽ định cấu hình DNS tổ chức của mình để trỏ tới bộ cân bằng tải này khi người dùng lưu trữ <http://myapp.com>. Giờ đây, việc tự mình thiết lập bộ cân bằng tải đó là một công việc tẻ nhạt và tôi có thể phải thực hiện việc đó trong môi trường cục bộ hoặc tại chỗ của mình. Tuy nhiên, nếu tôi tình cờ sử dụng CloudPlatform được hỗ trợ, như Google Cloud Platform, tôi có thể tận dụng các chức năng cân bằng tải gốc của nền tảng đám mây để thiết lập tính năng này. Một lần nữa, bạn không cần phải thiết lập thủ công, Kubernetes sẽ thiết lập nó cho bạn. Kubernetes có tích hợp sẵn với các nền tảng đám mây được hỗ trợ.

```
dịch vụ-định nghĩa.yml  
apiVersion: v1 loại:  
Dịch vụ  
metadata:  
    Tên: giao diện người dùng  
  
    móng sát mới:  
        loại: NLoadBalancer công:  
            - Công dịch: 80  
              công: 80  
  
        bộ chọn:  
            ứng dụng: myapp  
            Kiểu: giao diện người dùng
```

```
> kubectl tạo -f dịch vụ-định nghĩa.yml  
dịch vụ "front-end" đã được tạo
```

```
> kubectl nhận dịch vụ
```

| TÊN                  | Kiểu      | CLOUD-IP                        | (Các) CÔNG IP BÊN NGOÀI | TUỔI    |
|----------------------|-----------|---------------------------------|-------------------------|---------|
| giao diện người dùng | ClusterIP | 10.96.0.1                       | loaBalancer 443/TCP     | 16 ngày |
| dung kubernetes      |           | 10.106.127.123 <Đang chờ xử lý> | 80/TCP                  | 2m      |

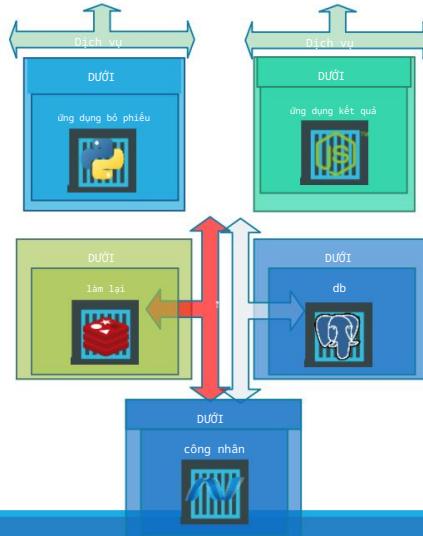


Trong bài giảng này, chúng ta sẽ thử tìm hiểu kiến trúc Microservices bằng cách sử dụng một ứng dụng web đơn giản. Sau đó, chúng tôi sẽ thử và triển khai ứng dụng web này trên nhiều nền tảng Kubernetes khác nhau.

## Ứng dụng bô phiếu mău

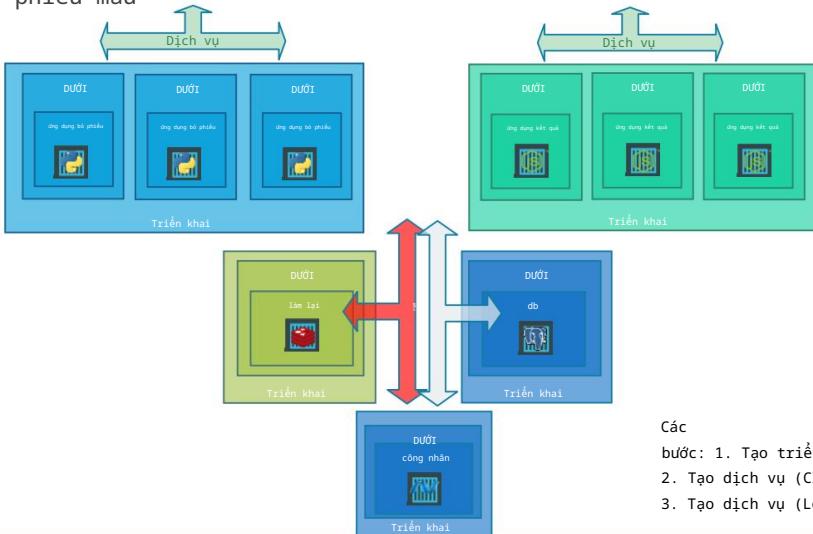
Mục  
tiêu: 1. Triển khai ứng dụng bô phiếu  
chứa 2. Kích hoạt kết nối 3.  
Truy cập bên ngoài

Các  
bước: 1. Triển khai  
POD 2. Tạo dịch vụ (ClusterIP)  
3. Tạo dịch vụ (LoadBalancer)



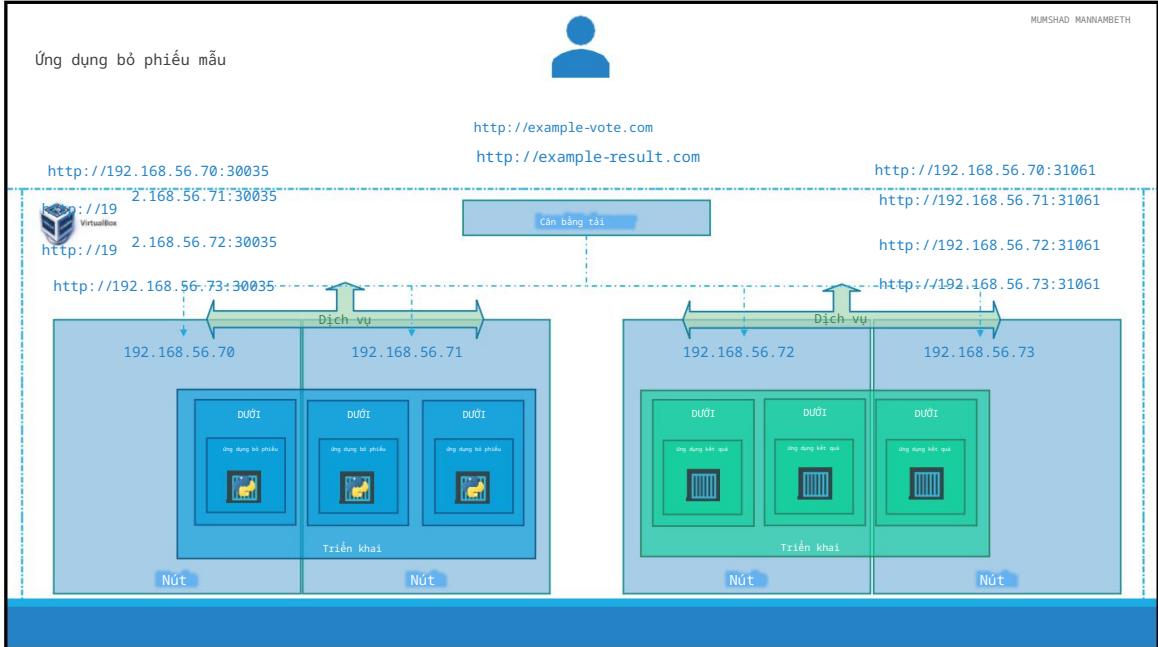
Đây là những gì chúng ta đã thấy trong bản demo trước. Chúng tôi đã triển khai POD và dịch vụ để kiểm soát mọi việc thực sự đơn giản. Nhưng điều này có những thách thức riêng của nó. Việc triển khai POD không giúp chúng tôi mở rộng quy mô ứng dụng của mình một cách dễ dàng. Ngoài ra, nếu POD bị lỗi thì nó sẽ không tự động sao lưu hoặc triển khai POD mới. Chúng ta có thể triển khai nhiều POD lần lượt, nhưng có nhiều cách dễ dàng hơn để mở rộng quy mô bằng cách sử dụng ReplicaSets và Deployments như chúng ta đã thảo luận trong các bài giảng.

## Ứng dụng bô phiếu mẫu

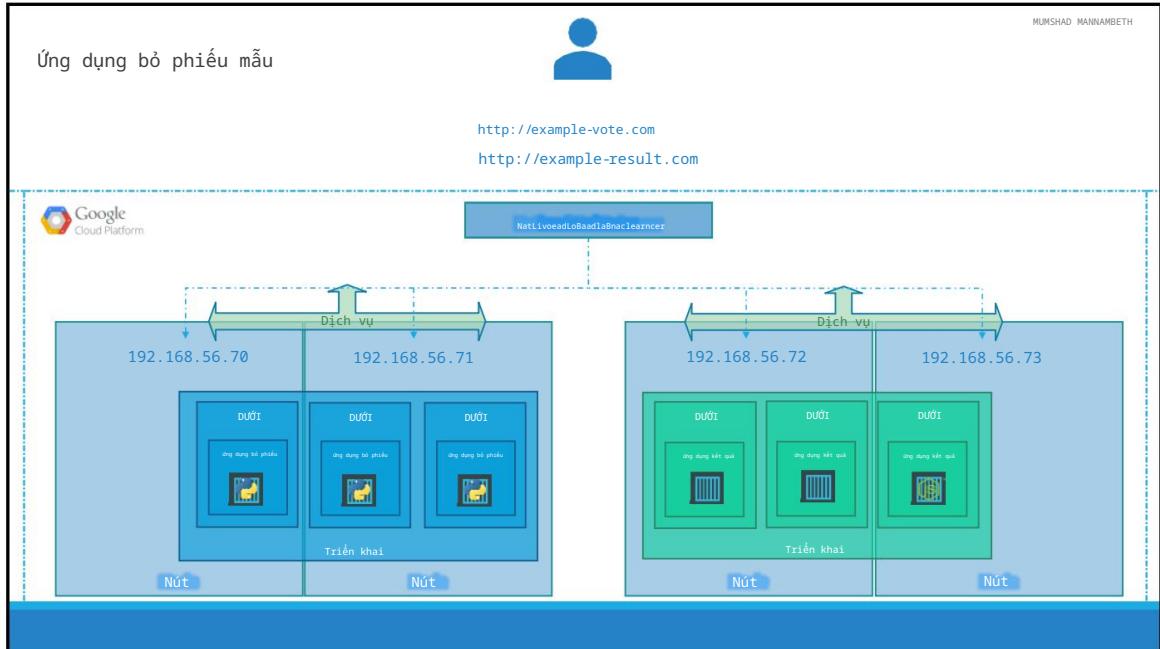


- Các bước:
1. Tạo triển khai
  2. Tạo dịch vụ (ClusterIP)
  3. Tạo dịch vụ (LoadBalancer)

Bây giờ chúng ta hãy ứng biến thiết lập của mình bằng cách sử dụng Triển khai. Chúng tôi đã chọn triển khai thay vì ReplicaSets vì Triển khai tự động tạo các bộ bản sao theo yêu cầu và nó có thể giúp chúng tôi thực hiện các bản cập nhật luân phiên và khôi phục, v.v. Triển khai là cách tốt nhất. Vì vậy, chúng tôi bổ sung thêm nhiều POD cho ứng dụng bô phiếu và ứng dụng kết quả của ứng dụng giao diện người dùng bằng cách tạo một bản triển khai với 3 bản sao. Chúng tôi cũng đóng gói cơ sở dữ liệu và công nhân trong quá trình triển khai. Chúng ta hãy nhìn vào điều đó bây giờ.



Bây giờ chúng ta hãy tập trung vào các ứng dụng giao diện người dùng - ứng dụng bỏ phiếu và ứng dụng kết quả. Chúng tôi biết rằng các POD này được lưu trữ trên các Nút khác nhau. Các dịch vụ có loại NodePort giúp nhận lưu lượng truy cập trên các cổng trên Nút và định tuyến cũng như cân bằng tải chúng đến các POD thích hợp. Nhưng bạn sẽ cung cấp địa chỉ IP nào cho người dùng cuối của mình để truy cập ứng dụng. Bạn có thể truy cập bất kỳ ứng dụng nào trong số hai ứng dụng này bằng IP của bất kỳ Nút nào và cổng cao cấp mà dịch vụ được cung cấp. Đó sẽ là sự kết hợp 4 IP và cổng cho ứng dụng bỏ phiếu và 4 kết hợp IP và cổng cho ứng dụng kết quả. Nhưng đó không phải là điều người dùng cuối mong muốn. Họ cần một URL duy nhất như example-vote.com hoặc example-result.com. Một cách để đạt được điều này trong thiết lập VirtualBox hiện tại của tôi là tạo một VM mới cho mục đích Cân bằng tải, đồng thời cài đặt và định cấu hình bộ cân bằng tải phù hợp trên đó như HAProxy hoặc NGINX, v.v. Sau đó, bộ cân bằng tải sẽ định tuyến lại lưu lượng truy cập đến các nút bên dưới và sau đó thông qua POD để phục vụ người dùng.



Bây giờ, việc thiết lập tất cả cân bằng tải bên ngoài đó có thể là một công việc tẻ nhạt. Tuy nhiên, nếu tôi đang sử dụng nền tảng đám mây được hỗ trợ như Google Cloud, tôi có thể tận dụng bộ cân bằng tải gốc để thực hiện cấu hình đó cho mình. Kubernetes có hỗ trợ để tự động thực hiện việc đó cho chúng tôi. Tất cả những gì bạn cần làm là đặt Loại dịch vụ cho các dịch vụ giao diện người dùng thành LoadBalancer. Chỉ nhớ điều này, hoạt động với các nền tảng đám mây được hỗ trợ. Nếu bạn đặt Loại dịch vụ thành LoadBalancer trong môi trường không được hỗ trợ như VirtualBox thì điều đó sẽ chỉ được coi là như thể bạn đặt nó thành NodePort, nếu các dịch vụ được xuất trên cổng cao cấp trên Nút. Thiết lập tương tự như chúng tôi đã có trước đó. Chúng ta hãy xem cách chúng ta có thể đạt được điều này trên Google Cloud Platform.

## Phần kết luận

Tổng quan về Kubernetes  
 Container - Docker  
 Dàn nhạc container?  
 Demo - Thiết lập Kubernetes  
 Khái niệm Kubernetes - POD | Bản sao | Triển khai | Dịch vụ  
 Kết nối mạng trong Kubernetes  
 Quản lý Kubernetes - Kubectl  
 Tệp định nghĩa Kubernetes- YAML  
 Kubernetes trên đám mây - AWS/GCP

Chúng ta đã kết thúc khóa học Kubernetes dành cho người mới bắt đầu. Tôi hy vọng tôi đã đề cập đủ chủ đề để giúp bạn bắt đầu với Kubernetes. Chúng tôi đã bắt đầu với Container và Docker và tìm hiểu cách điều phối vùng chứa là gì. Chúng tôi đã xem xét nhiều tùy chọn có sẵn để thiết lập Kubernetes. Chúng ta đã tìm hiểu một số khái niệm như POD, Bộ bản sao, Triển khai và Dịch vụ. Chúng tôi cũng đã xem xét Mạng trong Kubernetes.

Chúng tôi cũng đã dành chút thời gian để làm việc với các lệnh kubectl và tệp định nghĩa Kubernetes. Tôi hy vọng bạn có đủ kinh nghiệm thực tế trong việc phát triển tệp định nghĩa Kubernetes. Và cuối cùng, chúng ta cũng đã biết cách triển khai một ứng dụng vi dịch vụ mẫu trên Kubernetes trên Google Cloud Platform.

Tôi sẽ bổ sung thêm các chủ đề khác vào khóa học trong thời gian tới. Vì vậy hãy chú ý đến thông báo từ tôi. Trong trường hợp bạn cần tôi đề cập đến bất kỳ chủ đề nào khác, vui lòng gửi tin nhắn cho tôi hoặc gửi câu hỏi và tôi sẽ cố gắng hết sức để thêm một bài giảng mới về chủ đề đó.

Chúng tôi cũng có một bài tập như một phần của khóa học này. Vì vậy, nếu bạn có thời gian, vui lòng tiếp tục phát triển giải pháp của mình và nộp bài tập. Hãy thoải mái xem lại bài tập của học sinh khác và chia sẻ quan điểm của bạn về chúng.

Thật tuyệt vời khi có bạn tham gia khóa học này. Tôi hy vọng bạn đã có đủ kiến thức và kinh nghiệm để bắt đầu với Kubernetes trong công việc của mình hoặc nói cách khác và tôi chúc bạn may mắn trong Hành trình Kubernetes của mình. Hẹn gặp lại các bạn trong khóa học tiếp theo, cho đến lúc đó Good Bye!