BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

ĐỀ CƯƠNG

CHUYÊN ĐỀ AN TOÀN HỆ THỐNG THÔNG TIN

Nghiên cứu giải pháp đảm bảo an toàn dữ liệu trên Kubernetes

Ngành: An toàn thông tin

Sinh viên thực hiện:
Nguyễn Anh Tuấn
Mã sinh viên: AT160258
Đặng Sơn Hà
Mã sinh viên: AT160220

Người hướng dẫn:
TS. Nguyễn Mạnh Thắng
Khoa An toàn thông tin - Học viện Kỹ thuật mật mã

Hà Nội, 2022

Mục lục

Lời cảm ơn

Nhóm chúng em xin chân thành cảm ơn các thầy cô trường Học viện Kỹ thuật Mật mã nói chung, quý thầy cô của khoa An toàn thông tin nói riêng đã tận tình dạy bảo, truyền đạt kiến thức cho chúng em trong suốt quá trình học.

Kính gửi đến Thầy Nguyễn Mạnh Thắng lời cảm ơn chân thành và sâu sắc nhất, cảm ơn thầy đã tận tình theo sát, chỉ bảo và hướng dẫn cho nhóm em trong quá trình thực hiện đề tài này. Thầy không chỉ hướng dẫn chúng em những kiến thức chuyên ngành, mà còn giúp chúng em học thêm những kĩ năng mềm, tinh thần học hỏi, thái độ khi làm việc nhóm.

Trong quá trình tìm hiểu nhóm chúng em xin cảm ơn các bạn sinh viên đã góp ý, giúp đỡ và hỗ trợ nhóm em rất nhiều trong quá trình tìm hiểu và làm đề tài.

Do kiến thức còn nhiều hạn chế nên không thể tránh khỏi những thiếu sót trong quá trình làm đề tài.Chúng em rất mong nhận được sự đóng góp ý kiến của quý thầy cô để đề tài của chúng em đạt được kết quả tốt hơn.

Chúng em xin chân thành cảm ơn!

Lời mở đầu

Trong những năm gần đây, Kubernetes đã trở thành một trong những công cụ quản lý container phổ biến nhất. Điều này là do sự phát triển mạnh mẽ của Kubernetes, cũng như sự phổ biến của container. Trong các doanh nghiệp hiện nay, Kubernetes là một phần không thể thiếu được sử dụng để quản lý các ứng dụng container, cung cấp các dịch vụ như load balancing, autoscaling, logging, monitoring, backup và restore, cũng như quản lý các tài nguyên của các ứng dụng như CPU, Memory, Storage, Network. Tuy nhiên, việc sử dụng Kubernetes cũng có những hạn chế, đặc biệt là trong việc quản lý các cấu hình nhạy cảm của ứng dụng và các tài nguyên của ứng dụng. Vì vậy, trong đề tài này, chúng ta sẽ tìm hiểu về Vault, một công cụ quản lý secret phổ biến nhất hiện nay, được phát triển bởi HashiCorp.

Vault được sử dụng để quản lý các secret như username, password, token, key, certificate, API key, SSH key, license key, database connection và các thông tin khác. Vì vậy trong tương lai, Vault sẽ là một trong những công cụ quản lý secret không thể thiếu trong Kubernetes.

Với những lí do trên, bài toán đặt ra ở đây là làm sao để triển khai Vault trên cụm Kubernetes. Vậy nên, đề tài "Nghiên cứu giải pháp đảm bảo an toàn dữ liệu trên K8S" được thực hiện trong báo cáo có ý nghĩa khoa học và mang tính thực tiễn cao.

Chương 1

Giới thiệu về Kurbenetes

1.1 Tổng quan về Kurbenetes

Kubernetes hoặc k8s là một nền tảng mã nguồn mở giúp tự động hóa việc quản lý, mở rộng và triển khai ứng dụng dưới dạng container. Kubernetes còn được gọi là Container Orchestration Engine (hiểu nôm na là công cụ điều phối container). Kubernetes loại bỏ rất nhiều các quy trình thủ công liên quan đến việc triển khai và mở rộng các containerized applications. Các ứng dụng có thể khác nhau về kích thước: từ 1 cho đến hàng nghìn server. Với Kubernetes chúng ta có thể phát triển application một cách linh hoạt và đáng tin cậy.

Trách nhiệm chính của Kubernetes là container orchestration (dịch ra có nghĩa điều phối container). Kubernetes đảm bảo rằng tất cả container được lên lịch chạy trên các cụm server (cluster) (server ở đây có thể là physical machine hoặc virtual machine). Ngoài ra, Kubernetes còn có chức năng theo dõi hoạt động của từng container, mở rộng các containers và quản lý tình trạng của các containers theo thời gian. Khi một container nào đó gặp trục trặc, dừng hoạt động thì Kubernetes sẽ thay thế container đó.

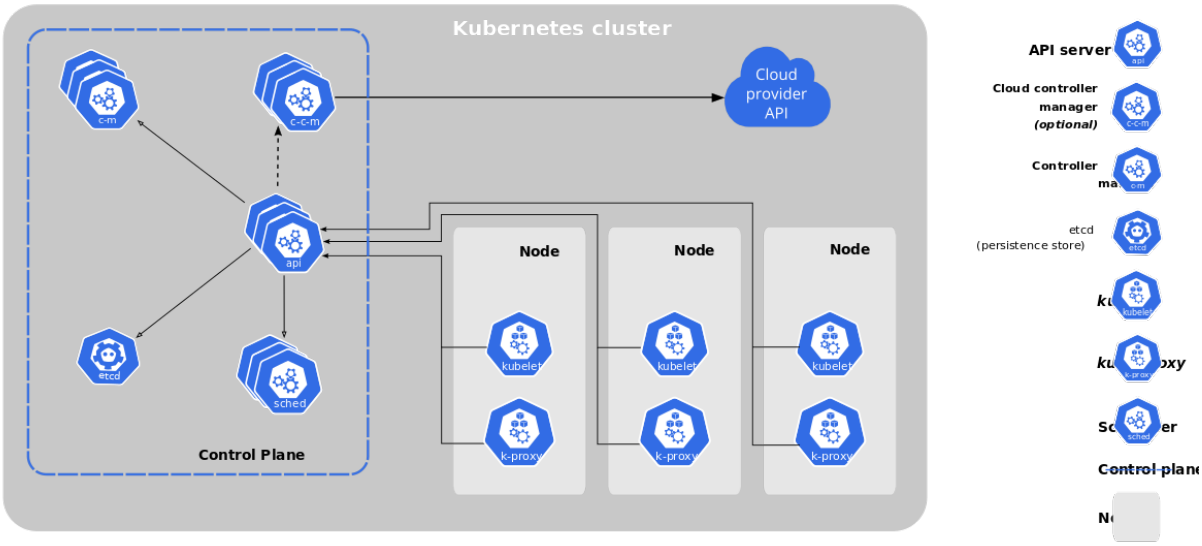
Kubernetes ban đầu được các kỹ sư Google phát triển vào năm 2014. Và Google cũng là một trong những cái tên tiên phong đóng góp cho công cuộc phát triển công nghệ Linux container.

Kubernetes ban đầu được các kỹ sư Google phát triển vào năm 2014. Và Google cũng là một trong những cái tên tiên phong đóng góp cho công cuộc phát triển công nghệ Linux container.

Red Hat là một trong những công ty đầu tiên hợp tác với Google trong dự án Kubernetes trước khi Kubernetes được ra mắt, và trở thành nhà tài trợ lớn thứ 2 cho dự án ngay từ những ngày đầu. Và Google đã tặng lại dự án Kubernetes cho Cloud Native Computing Foundation (CNCF - thành lập năm 2015). Kubernetes đã trở thành một project của tổ chức này cho đến thời điểm hiện tại. Năm 2018, Kubernetes Project đạt vị trí thứ 9 về số lượt commit trên Github.

1.1.1 Các thành phần trong kubernetes

Khi bạn triển khai Kubernetes, bạn sẽ nhận được một Cluster (cụm máy chủ).



Hình 1.1: Sơ đồ của một cụm Kubernetes.

Một Cluster Kubernetes bao gồm một tập hợp các máy worker, được gọi là Node, chạy các ứng dụng được đóng gói. Mỗi Cluster có ít nhất một Node.

(Các) Node lưu trữ các Pod - Một nhóm gồm một hoặc nhiều containers được triển khai cho single Node, nhóm này chia sẻ chung không gian lưu trữ, IP address, hostname, và những nguồn khác.

Các Control Plane quản lý các Node và các Pod trong Cluster. Trong môi trường production, Control Plane thường chạy trên nhiều máy tính và một Cluster thường chạy nhiều Node, mang lại khả năng chịu lỗi và tính sẵn sàng cao.

Trên đây là sơ đồ của một cụm Kubernetes với tất cả các thành phần được gắn với nhau.

Các thành phần Control Plane (Master)

Các thành phần Control Plane đưa ra các quyết định chung về Cluster (ví dụ: lập lịch), cũng như phát hiện và phản hồi các sự kiện của Cluster.

Các thành phần Control Plane có thể được chạy trên bất kỳ máy nào trong Cluster. Tuy nhiên, để đơn giản, thiết lập tập lệnh thường khởi động tất cả các thành phần Control Plane trên cùng một máy và không chạy các container của người dùng trên máy này.

Đối với cluster nhỏ , Master có thể chạy trên một Node, nhưng trong một cluster lớn, để đảm bảo tính khả dụng (trong tiếng anh là High-Availability) thì Master có thể được chạy trên nhiều Node. (Tính khả dụng có nghĩa là Khi mà một Node trong cluster dừng hoạt động thì hệ thống vẫn duy trì như không có gì xảy ra). Master sẽ bao gồm 5 thành phần chính sau:

- **kube-apiserver:**

  API Server là một thành phần của Control Plane. Đúng theo tên gọi, đây chính là server cung cấp REST API cho Kubernetes Cluster. Nó có nhiệm vụ đặt Pod vào

Node, đồng bộ hoá thông tin của Pod bằng REST API tiếp nhận cài đặt, xác thực và thiết lập cho các objects pod/service/replicationController.

- **Etcd:**

  Etcd là Kho lưu trữ giá trị khóa nhất quán và có tính khả dụng cao được sử dụng làm kho dự phòng để lưu trữ toàn bộ cấu hình, trạng thái và metadata của Kubernetes Cluster.

  Trong các cluster nhỏ, Etcd có thể chạy trên cùng một Node với các thành phần khác. Nhưng trong các cluster lớn, Etcd có thể chạy dự phòng trên nhiều Node để đảm bảo tính khả dụng của toàn hệ thống. Nếu cụm Kubernetes của bạn sử dụng Etcd làm nơi lưu trữ sao lưu, hãy đảm bảo rằng bạn có kế hoạch sao lưu cho những dữ liệu đó.

- **kube-controller-manager:**

  Kube Controller Manage một tập hợp các controller khác nhau để chạy các bộ điều khiển quy trình theo dõi các cập nhật trạng thái của Kubernetes Cluster thông qua API và thực hiện các thay đổi đối với Cluster sao cho phù hợp.

  Về mặt logic, mỗi bộ điều khiển là một quy trình riêng biệt, nhưng để giảm độ phức tạp, tất cả chúng đều được biên dịch thành một tệp nhị phân duy nhất và chạy trong một quy trình duy nhất.

  Một số loại bộ điều khiển có thể kể đến:

  - Node controller: Chịu trách nhiệm thông báo và phản hồi khi các Node gặp sự cố.
  - Job controller: Kiểm soát các tác vụ được lập lịch, sau đó tạo các Pod để chạy các tác vụ đó đến khi hoàn thành.
  - Endpoints controller: Điền khiển đối tượng Điểm cuối (nghĩa là tham gia Dịch vụ & Nhóm).
  - Service Account & Token controllers: Tạo tài khoản mặc định và mã thông báo truy cập API cho không gian tên mới.

- **cloud-controller-manager:**

  Là một tập hợp các logic dành riêng cho đám mây (GCP, AWS, Azure) cho phép bạn liên kết Kubernetes Cluster với API của nhà cung cấp đám mây và tách các thành phần tương tác với nền tảng đám mây đó khỏi các thành phần chỉ tương tác với cụm của bạn. Cloud-controller-manager chỉ chạy các bộ điều khiển dành riêng cho nhà cung cấp dịch vụ đám mây của bạn. Nếu bạn đang chạy Kubernetes tại cơ sở của riêng mình hoặc trong môi trường học tập bên trong PC của riêng bạn, cụm không có trình quản lý bộ điều khiển đám mây.

  Cũng như với kube-controller-manager, cloud-controller-manager kết hợp một số vòng điều khiển độc lập về mặt logic thành một tệp nhị phân duy nhất mà bạn chạy như một quy trình duy nhất. Bạn có thể chia tỷ lệ theo chiều ngang (chạy nhiều hơn một bản sao) để cải thiện hiệu suất hoặc để giúp tăng khả năng chịu lỗi.

  Các bộ điều khiển sau có thể có các phần phụ thuộc của nhà cung cấp dịch vụ đám mây:

- Node controller: Để kiểm tra nhà cung cấp đám mây để xác định xem một Node đã bị xóa trong đám mây sau khi nó ngừng phản hồi hay chưa.
- Route controller: Để thiết lập định tuyến trong cơ sở hạ tầng đám mây cơ bản.
- Service controller: Để tạo, cập nhật và xóa bộ cân bằng tải của nhà cung cấp dịch vụ đám mây.

- **kube-scheduler:**

  Sử dụng Kubernetes API để tìm, theo dõi các Pod chưa được lên lịch hoặc mới được tạo nhưng không được chỉ định Node. Sau đó, scheduler sẽ đặt các Pod này vào các Node dựa trên tài nguyên và các ràng buộc khác được định nghĩa trong manifest file của Pod. Scheduler sẽ cố gắng đảm bảo rằng các Pod của cùng một application sẽ được phân phối trên các Node khác nhau để đảm bảo tính khả dụng.

  Các yếu tố được tính đến để đưa ra quyết định lập lịch bao gồm: yêu cầu tài nguyên cá nhân và tập thể, các ràng buộc về phần cứng / phần mềm / chính sách, thông số kỹ thuật về mối quan hệ và chống mối quan hệ, vị trí dữ liệu, can thiệp giữa khối lượng công việc và thời hạn.

Các thành phần Worker

Có nhiệm vụ xử lý khối lượng công việc của application trong cluster, duy trì các nhóm đang chạy và cung cấp môi trường runtime cho Kubernetes. Worker sẽ bao gồm 3 thành phần chính sau:

- **kubelet:**

  Kubelet chạy trên mỗi Worker Node. Nó đảm bảo rằng các container đang chạy trong một Pod và có trách nhiệm giám sát giao tiếp với master node và quản lý các Pod. Kubelet sử dụng CRI (Container Runtime Interface - Giao diện thời gian chạy vùng chứa) để giao tiếp với container runtime trên cùng một Node đó. Kubelet không quản lý các vùng chứa không được tạo bởi Kubernetes.

- **kube-proxy:**

  Là một proxy mạng chạy trên mỗi Node trong Cluster. Nó có trách nhiệm quản lý, duy trì các quy tắc mạng trên mỗi Node. Các quy tắc mạng này cho phép giao tiếp mạng với Pod của bạn từ các phiên mạng bên trong hoặc bên ngoài Cluster. kube-proxy sử dụng lớp lọc gói của hệ điều hành nếu có và nó có sẵn. Nếu không, kube-proxy sẽ tự chuyển tiếp lưu lượng.

- **Container Runtime:**

  Là phần mềm chịu trách nhiệm chạy các container. Kubernetes hỗ trợ một số container runtime như: Docker, containerd, CRI-O và bất kỳ triển khai nào của Kubernetes CRI (Container Runtime Interface).

1.2 Các đối tượng trong Kubernetes

1.2.1 Giới thiệu các đối tượng trong Kubernetes

Các đối tượng Kubernetes là các thực thể liên tục trong hệ thống Kubernetes. Kubernetes sử dụng các thực thể này để biểu thị trạng thái của cụm của bạn. Cụ thể, có thể

mô tả:

- Những ứng dụng được đóng gói nào đang chạy (và chạy trên các node nào).
- Các tài nguyên có sẵn cho các ứng dụng đó.
- Các chính sách xung quanh cách các ứng dụng đó hoạt động, chẳng hạn như chính sách khởi động lại, nâng cấp và khả năng chịu lỗi.

Đối tượng Kubernetes là một "bản ghi mục đich" - khi tạo đối tượng, hệ thống Kubernetes sẽ liên tục hoạt động để đảm bảo đối tượng đó tồn tại. Bằng cách tạo một đối tượng, bạn đang nói cho hệ thống Kubernetes biết bạn muốn khối lượng công việc của cụm trông như thế nào, đây là trạng thái mong muốn của cụm của bạn.

Để làm việc với các đối tượng Kubernetes dù là tạo, sửa đổi hay xóa chúng, sẽ cần sử dụng Kubernetes API. Ví dụ: khi bạn sử dụng giao diện dòng lệnh `kubectl`, CLI sẽ thực hiện các lệnh gọi API Kubernetes cần thiết cho bạn. Bạn cũng có thể sử dụng Kubernetes API trực tiếp trong các chương trình của riêng mình bằng cách sử dụng một trong các Thư viện máy khách.

Đối tượng spec và status

Hầu hết mọi đối tượng Kubernetes đều bao gồm hai trường đối tượng lồng nhau chi phối cấu hình của đối tượng: đối tượng `spec` và đối tượng `status`. Đối với mỗi đối tượng,phải đặt `spec` khi khởi tạo, cung cấp mô tả về các đặc điểm muốn tài nguyên có: được gọi là trạng thái mong muốn của đối tượng.

`status` mô tả trạng thái hiện tại của đối tượng, được cung cấp và cập nhật bởi hệ thống Kubernetes và các thành phần của nó. Control plane Kubenertes liên tục và tích cực quản lý trạng thái thực tế của mọi đối tượng để phù hợp với trạng thái mong muốn đã cung cấp.

Ví dụ: trong Kubernetes, Deployment là một đối tượng có thể đại diện cho một ứng dụng đang chạy trên cụm của bạn đảm nhận việc deploy và quản lý ứng dụng. Khi bạn tạo Deployment, bạn có thể đặt `spec` cho Deployment để chỉ định rằng bạn muốn chạy ba bản sao của ứng dụng. Hệ thống Kubernetes đọc thông số kỹ thuật Deployment và bắt đầu ba phiên bản của ứng dụng bạn mong muốn, cập nhật trạng thái để phù hợp với thông số kỹ thuật của bạn. Nếu bất kỳ trường hợp nào trong số đó không thành công (thay đổi trạng thái), hệ thống Kubernetes sẽ phản hồi sự khác biệt giữa thông số kỹ thuật và trạng thái bằng cách thực hiện chỉnh sửa. Lúc này,hệ thống Kubenertes bắt đầu một trường hợp thay thế khác.

Mô tả một đối tượng Kubernetes

Khi tạo một đối tượng trong Kubernetes phải cung cấp thông số đối tượng mô tả trạng thái mong muốn của nó, cũng như một số thông tin cơ bản về đối tượng (chẳng hạn như tên). Khi bạn sử dụng API Kubernetes để tạo đối tượng (trực tiếp hoặc thông qua `kubectl`), yêu cầu API đó phải bao gồm thông tin đó dưới dạng JSON trong phần thân yêu cầu. Thông thường, bạn cung cấp thông tin cho `kubectl` trong tệp .yaml. `kubectl` chuyển đổi thông tin thành JSON khi thực hiện yêu cầu API.

Đây là một tệp .yaml ví dụ hiển thị các trường bắt buộc và thông số đối tượng cho một Kubernetes Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  spec:
    selector:
      matchLabels:
        app: nginx
    replicas: 2 #tells deployment to run 2 pods matching the template
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
         - name: nginx
           image: nginx:1.14.2
           ports:
            - containerPort: 80
```

Một cách để tạo Deployment bằng tệp .yaml như trên là sử dụng lệnh `kubectl apply` trong giao diện dòng lệnh `kubectl`, chuyển tệp .yaml dưới dạng đối số. Đây là một ví dụ:

```
# kubectl apply -f https://k8s.io/examples/application/deployment.yaml
```

Đầu ra tương tự như thế này:

```
# deployment.apps/nginx-deployment created
```

Phần bắt buộc

Trong tệp .yaml dành cho đối tượng Kubernetes muốn tạo, sẽ cần đặt giá trị cho các trường sau:

- `apiVersion`- Phiên bản Kubernetes API nào sử dụng để tạo đối tượng này.
- `kind`- Loại đối tượng bạn muốn tạo.
- `metadata`- Dữ liệu giúp nhận dạng duy nhất đối tượng, bao gồm `name` chuỗi `UID`, và tùy chọn `namespace`.
- `spec`- Trạng thái bạn mong muốn cho đối tượng

Định dạng chính xác của đối tượng `spec` là khác nhau đối với mọi đối tượng Kubernetes và chứa các trường lồng nhau dành riêng cho đối tượng đó.

1.2.2 Quản lý các đối tượng trong Kubernetes

Công cụ dòng lệnh `kubectl` hỗ trợ một số cách khác nhau để tạo và quản lý các đối tượng Kubernetes. Tài liệu này cung cấp một cái nhìn tổng quan về các cách tiếp cận khác nhau.

Kỹ thuật quản lý	Hoạt động trên	Môi trường đề xuất
Dòng lệnh bắt buộc	Đối tượng trực tiếp	Dự án phát triển
Cấu hình đối tượng bắt buộc	Các tệp riêng lẻ	Dự án sản xuất
Cấu hình đối tượng khai báo	Thư mục của tập tin	Dự án sản xuất

Kỹ thuật quản lý: Một đối tượng Kubernetes chỉ nên được quản lý bằng một kỹ thuật. Các kỹ thuật trộn lẫn và kết hợp cho cùng một đối tượng dẫn đến hành vi không xác định.

Dòng lệnh bắt buộc

Khi sử dụng các dòng lệnh bắt buộc, người dùng thao tác trực tiếp trên các đối tượng trực tiếp trong một cụm. Người dùng cung cấp các hoạt động cho lệnh `kubectl` dưới dạng đối số hoặc cờ.

Đây là cách được đề xuất để bắt đầu hoặc chạy tác vụ một lần trong cụm. Vì kỹ thuật này hoạt động trực tiếp trên các đối tượng trực tiếp nên nó không cung cấp lịch sử của các cấu hình trước đó.

Ví dụ: Chạy một phiên bản nginx container bằng cách tạo một đối tượng Deployment:

```
# kubectl create deployment nginx -image nginx
```

So sánh:

- Ưu điểm so với cấu hình đối tượng:
  - Các lệnh được thể hiện dưới dạng một từ hành động.
  - Các lệnh chỉ yêu cầu một bước duy nhất để thực hiện các thay đổi đối với cụm.
- Nhược điểm so với cấu hình đối tượng:
  - Các lệnh không tích hợp với các quy trình xem xét thay đổi.
  - Các lệnh không cung cấp dấu vết kiểm tra liên quan đến các thay đổi.
  - Các lệnh không cung cấp nguồn bản ghi ngoại trừ những gì đang hoạt động.
  - Các lệnh không cung cấp mẫu để tạo đối tượng mới.

Cấu hình đối tượng bắt buộc

Trong cấu hình đối tượng bắt buộc, lệnh `kubectl` chỉ định thao tác (tạo, thay thế, v.v.), các cờ tùy chọn và ít nhất một tên tệp. Tệp được chỉ định phải chứa định nghĩa đầy đủ về đối tượng ở định dạng YAML hoặc JSON.

Ví dụ: Tạo các đối tượng được xác định trong tệp cấu hình:

```
# kubectl create -f nginx.yaml
```

Xóa các đối tượng được xác định trong hai tệp cấu hình:

```
# kubectl delete -f nginx.yaml -f redis.yaml
```

Cập nhật các đối tượng được xác định trong tệp cấu hình bằng cách ghi đè lên cấu hình trực tiếp:

```
# kubectl replace -f nginx.yaml
```

So sánh:

- Ưu điểm so với dòng lệnh bắt buộc:
  - Cấu hình đối tượng có thể được lưu trữ trong một hệ thống kiểm soát nguồn như Git.
  - Cấu hình đối tượng có thể tích hợp với các quy trình như xem xét các thay đổi trước khi đẩy và theo dõi kiểm tra.
  - Cấu hình đối tượng cung cấp một mẫu để tạo các đối tượng mới.
- Nhược điểm so với dòng lệnh bắt buộc:
  - Cấu hình đối tượng yêu cầu hiểu biết cơ bản về lược đồ đối tượng.
  - Cấu hình đối tượng yêu cầu thêm bước ghi tệp YAML.
- Ưu điểm so với cấu hình đối tượng khai báo:
  - Hành vi cấu hình đối tượng bắt buộc đơn giản hơn và dễ hiểu hơn.
  - Kể từ phiên bản Kubernetes 1.5, cấu hình đối tượng bắt buộc đã hoàn thiện hơn.
- Nhược điểm so với cấu hình đối tượng khai báo:
  - Cấu hình đối tượng bắt buộc hoạt động tốt nhất trên tệp chứ không phải thư mục.
  - Các bản cập nhật cho các đối tượng trực tiếp phải được phản ánh trong các tệp cấu hình, nếu không chúng sẽ bị mất trong lần thay thế tiếp theo.

Cấu hình đối tượng khai báo

Khi sử dụng cấu hình đối tượng khai báo, người dùng thao tác trên tệp cấu hình đối tượng được lưu trữ cục bộ, tuy nhiên người dùng không xác định các thao tác được thực hiện trên tệp. Các hoạt động tạo, cập nhật và xóa được tự động phát hiện trên mỗi đối tượng bởi `kubectl`. Điều này cho phép làm việc trên các thư mục, nơi có thể cần các thao tác khác nhau cho các đối tượng khác nhau.

Lưu ý: Cấu hình đối tượng khai báo giữ lại các thay đổi do người viết khác thực hiện, ngay cả khi các thay đổi không được hợp nhất trở lại tệp cấu hình đối tượng. Điều này có thể thực hiện được bằng cách sử dụng thao tác `patch` API để chỉ ghi những điểm khác biệt được quan sát, thay vì sử dụng thao tác `replace` API để thay thế toàn bộ cấu hình đối tượng.

Ví dụ: Xử lý tất cả các tệp cấu hình đối tượng trong thư mục `configs` và tạo hoặc vá các đối tượng trực tiếp. Trước tiên, bạn có thể `diff` xem những thay đổi nào sẽ được thực hiện, sau đó áp dụng:

```
# kubectl diff -f configs/

# kubectl apply -f configs/
```

Xử lý đệ quy các thư mục:

```
# kubectl diff -R -f configs/

# kubectl apply -R -f configs/
```

So sánh:

- Ưu điểm so với cấu hình đối tượng bắt buộc:
  - Các thay đổi được thực hiện trực tiếp đối với các đối tượng trực tiếp được giữ lại, ngay cả khi chúng không được hợp nhất lại vào các tệp cấu hình.
  - Cấu hình đối tượng khai báo hỗ trợ tốt hơn cho hoạt động trên các thư mục và tự động phát hiện các loại hoạt động (tạo, vá, xóa) cho mỗi đối tượng.
- Nhược điểm so với cấu hình đối tượng bắt buộc:
  - Cấu hình đối tượng khai báo khó gỡ lỗi và hiểu kết quả hơn khi chúng không được như mong muốn.
  - Cập nhật một phần bằng cách sử dụng khác biệt tạo ra các hoạt động hợp nhất và vá lỗi phức tạp.

1.2.3 Tên và ID của các đối tượng trong kubernetes

Mỗi đối tượng trong cụm của có một Tên duy nhất cho loại tài nguyên đó. Mỗi đối tượng Kubernetes cũng có một UID duy nhất trên toàn bộ cụm.

Ví dụ: bạn chỉ có thể có một Pod được đặt tên `myapp-1234` trong cùng một không gian tên nhưng bạn có thể có một Pod và một Deployment được đặt tên riêng `myapp-1234`.

Đối với các thuộc tính không phải duy nhất do người dùng khai báo, Kubernetes cung cấp nhãn và chú thích.

Tên

là chuỗi do máy khách khai báo đề cập đến một đối tượng trong URL tài nguyên, chẳng hạn như `/api/v1/pods/some-name`.

Tại một thời điểm, chỉ một đối tượng của một loại nhất định có thể có một tên nhất định. Tuy nhiên, nếu bạn xóa đối tượng, bạn có thể tạo một đối tượng mới có cùng tên.

Lưu ý: Trong trường hợp các đối tượng đại diện cho một thực thể vật lý, chẳng hạn như Node đại diện cho máy chủ vật lý, khi máy chủ được tạo lại dưới cùng tên mà không xóa và tạo lại Node, Kubernetes coi máy chủ mới là máy chủ cũ, điều này có thể dẫn đến sự không thống nhất.

Dưới đây là bốn loại ràng buộc tên thường được sử dụng cho tài nguyên.

- **Tên miền phụ DNS**

  Hầu hết các loại tài nguyên yêu cầu một tên có thể được sử dụng làm tên miền phụ DNS như được định nghĩa trong RFC 1123 . Điều này có nghĩa là tên phải:

  – chứa không quá 253 ký tự
  – chỉ chứa các ký tự chữ và số viết thường, '-' hoặc '.'
  – bắt đầu bằng một ký tự chữ và số
  – kết thúc bằng một ký tự chữ và số

- **Tên nhãn RFC 1123**

  Một số loại tài nguyên yêu cầu tên của chúng tuân theo tiêu chuẩn nhãn DNS như được định nghĩa trong RFC 1123 . Điều này có nghĩa là tên phải:

  – chứa tối đa 63 ký tự
  – chỉ chứa các ký tự chữ và số viết thường hoặc '-'
  – bắt đầu bằng một ký tự chữ và số
  – kết thúc bằng một ký tự chữ và số

- **Tên đoạn đường dẫn**

  Một số loại tài nguyên yêu cầu tên của chúng để có thể được mã hóa an toàn dưới dạng đoạn đường dẫn. Nói cách khác, tên có thể không phải là "." hoặc ".." và tên không được chứa "/" hoặc "%".

  Đây là một bảng kê khai ví dụ cho một Pod có tên `nginx-demo`.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-demo
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

  Lưu ý: Một số loại tài nguyên có các hạn chế bổ sung về tên của chúng.

UID

Một chuỗi do hệ thống Kubernetes tạo để xác định duy nhất các đối tượng.

Mọi đối tượng được tạo trong toàn bộ thời gian tồn tại của cụm Kubernetes đều có một UID riêng biệt. Nó nhằm mục đích phân biệt giữa các lịch sử các lần xuất hiện của những thực thể tương đồng.

Kubernetes UID là các định danh duy nhất trên toàn cầu (còn được gọi là UUID). UUID được chuẩn hóa theo tiêu chuẩn ISO/IEC 9834-8 và ITU-T X.667.

1.3 Kiến trúc của cụm

1.3.1 Node

1.3.2 Giao tiếp giữa các Node và Control Plane

Các giao tiếp bao gồm các đường dẫn giao tiếp giữa máy chủ API và cụm Kubernetes. Mục đích là cho phép người dùng tùy chỉnh cài đặt của họ để củng cố cấu hình mạng sao cho cụm có thể chạy trên mạng không đáng tin cậy (hoặc trên các IP công khai hoàn toàn trên nhà cung cấp dịch vụ đám mây).

Node tới Control Plane

Kubernetes có mẫu API "hub-and-spoke". Tất cả việc sử dụng API từ các Node (hoặc Pod mà chúng chạy) sẽ chấm dứt tại máy chủ API. Không có thành phần Control plane nào khác được thiết kế để hiển thị các dịch vụ từ xa. Máy chủ API được định cấu hình để lắng nghe các kết nối từ xa trên cổng HTTPS an toàn (thường là 443) có bật một hoặc nhiều hình thức xác thực ứng dụng khách . Một hoặc nhiều hình thức ủy quyền phải được bật, đặc biệt nếu các yêu cầu ẩn danh hoặc mã thông báo tài khoản dịch vụ được cho phép.

Các Node phải được cung cấp chứng chỉ gốc công khai cho cụm sao cho chúng có thể kết nối an toàn với máy chủ API cùng với thông tin đăng nhập hợp lệ của ứng dụng khách. Một cách tiếp cận tốt là thông tin xác thực ứng dụng khách được cung cấp cho kubelet ở dạng chứng chỉ ứng dụng khách.

Các Pod muốn kết nối với máy chủ API có thể thực hiện việc này một cách an toàn bằng cách tận dụng tài khoản dịch vụ để Kubernetes sẽ tự động thêm chứng chỉ gốc công khai và mã thông báo hợp lệ vào nhóm khi nó được khởi tạo. Dịch vụ `kubernetes` (trong không gian tên `default`) được định cấu hình với một địa chỉ IP ảo được chuyển hướng (thông qua `kube-proxy`) đến điểm cuối HTTPS trên máy chủ API.

Các thành phần của control plane cũng giao tiếp với máy chủ API qua cổng an toàn.

Do đó, chế độ hoạt động mặc định cho các kết nối từ các Node và Pod chạy trên các Node đến Control plane được bảo mật theo mặc định và có thể chạy trên các mạng công cộng và/hoặc không đáng tin cậy.

Control plane tới node

Có hai đường dẫn giao tiếp chính từ Control plane (máy chủ API) đến các Node. Đầu tiên là từ máy chủ API đến quy trình kubelet chạy trên mỗi Node trong cụm. Thứ hai là từ máy chủ API đến bất kỳ Node, Pod hoặc Service nào thông qua chức năng proxy của máy chủ API.

- **API server đến kubelet**

  Các kết nối từ máy chủ API đến kubelet được sử dụng cho:

  – Tìm nạp nhật ký cho Pod.

  – Đính kèm (thường thông qua `kubectl`) vào các Pod đang chạy.

– Cung cấp chức năng chuyển tiếp cổng của kubelet.

Các kết nối này kết thúc tại điểm cuối HTTPS của kubelet. Theo mặc định, máy chủ API không xác minh chứng chỉ phục vụ của kubelet, điều này làm cho kết nối phải chịu các cuộc tấn công trung gian và không an toàn khi chạy trên các mạng công cộng và/hoặc không đáng tin cậy.

Để xác minh kết nối này, hãy sử dụng cờ `-kubelet-certificate-authority` để cung cấp cho máy chủ API gói chứng chỉ gốc nhằm sử dụng để xác minh chứng chỉ cung cấp của kubelet.

Nếu không thể, hãy sử dụng đường hầm SSH giữa máy chủ API và kubelet nếu cần để tránh kết nối qua mạng công cộng hoặc không đáng tin cậy.

Cuối cùng, xác thực và/hoặc ủy quyền Kubelet phải được bật để bảo mật API kubelet.

- **API server đến các Node, Pod hoặc Service**

  Các kết nối từ máy chủ API đến một Node, Pod hoặc Service mặc định là kết nối HTTP đơn giản và do đó không được xác thực cũng như không được mã hóa. Chúng có thể được chạy qua kết nối HTTPS an toàn bằng cách thêm tiền tố `https:` vào tên Node, Pod hoặc Service trong URL API, nhưng chúng sẽ không xác thực chứng chỉ do điểm cuối HTTPS cung cấp cũng như không cung cấp thông tin đăng nhập của khách hàng. Vì vậy, mặc dù kết nối sẽ được mã hóa nhưng nó sẽ không cung cấp bất kỳ đảm bảo nào về tính toàn vẹn. Các kết nối này hiện không an toàn để chạy qua các mạng công cộng hoặc không đáng tin cậy.

- **SSH tunnels**

  Kubernetes hỗ trợ các SSH tunnel để bảo vệ các đường dẫn giao tiếp của control plane đến Node. Trong cấu hình này, máy chủ API khởi tạo một SSH tunnel tới từng Node trong cụm (kết nối với máy chủ SSH đang nghe trên cổng 22) và chuyển tất cả lưu lượng dành cho kubelet, Node, Pod hoặc Service qua tunnel. Tunnel này đảm bảo rằng lưu lượng không bị lộ ra bên ngoài mạng mà các Node đang chạy.

  Lưu ý: SSH tunnel hiện không được dùng nữa, vì vậy không nên chọn sử dụng chúng trừ khi biết bản thân đang sử dụng với mục đích gì. Dịch vụ Konnectivity là sự thay thế cho kênh liên lạc này.

- **Dịch vụ Konnectivity**

  Để thay thế cho các SSH tunnel, dịch vụ Konnectivity cung cấp proxy cấp TCP cho mặt phẳng điều khiển để liên lạc theo cụm. Dịch vụ Konnectivity bao gồm hai phần: máy chủ Konnectivity trong mạng control plane và tác nhân Konnectivity trong mạng Node. Tác nhân Konnectivity khởi tạo kết nối đến máy chủ Konnectivity và duy trì kết nối mạng. Sau khi kích hoạt dịch vụ Konnectivity, tất cả lưu lượng truy cập từ control plane đến các Node đều đi qua các kết nối này.

1.3.3 Controllers

Trong chế tạo rô-bốt và tự động hóa, vòng lặp điều khiển là một vòng lặp không kết thúc điều chỉnh trạng thái của hệ thống.

Ví dụ: bộ điều chỉnh nhiệt trong phòng: Khi bạn đặt nhiệt độ, điều đó báo cho bộ điều chỉnh nhiệt về trạng thái mong muốn của bạn. Nhiệt độ phòng thực tế là trạng thái hiện tại. Bộ điều chỉnh nhiệt hoạt động để đưa trạng thái hiện tại đến gần trạng thái mong muốn hơn bằng cách bật hoặc tắt thiết bị.

Trong Kubernetes, bộ điều khiển là các vòng điều khiển theo dõi trạng thái trong cụm, sau đó thực hiện hoặc yêu cầu thay đổi nếu cần. Mỗi bộ điều khiển cố gắng di chuyển trạng thái cụm hiện tại đến gần trạng thái mong muốn hơn.

Kiểu điều khiển

Bộ điều khiển theo dõi ít nhất một loại tài nguyên Kubernetes. Các đối tượng này có một trường thông số đại diện cho trạng thái mong muốn. (Các) bộ điều khiển cho tài nguyên đó chịu trách nhiệm làm cho trạng thái hiện tại tiến gần hơn đến trạng thái mong muốn đó.

Bộ điều khiển có thể tự thực hiện hành động; thông thường hơn, trong Kubernetes, bộ điều khiển sẽ gửi tin nhắn đến máy chủ API có tác dụng phụ hữu ích.

- **Điều khiển thông qua máy chủ API**

  Công việc của bộ điều khiển là một ví dụ về bộ điều khiển tích hợp Kubernetes. Bộ điều khiển tích hợp quản lý trạng thái bằng cách tương tác với máy chủ API cụm.

  Công việc là một tài nguyên Kubernetes chạy một Pod, hoặc có thể là một số Pod, để thực hiện một tác vụ rồi dừng lại.

  (Sau khi được lên lịch , các đối tượng Pod trở thành một phần của trạng thái mong muốn cho một kubelet).

  Khi bộ điều khiển Công việc nhìn thấy một tác vụ mới, nó đảm bảo rằng, ở đâu đó trong cụm của bạn, các kubelet trên một nhóm các Node đang chạy đúng số lượng Pod để hoàn thành công việc. Bộ điều khiển công việc không tự chạy bất kỳ các Pod hoặc Container nào. Thay vào đó, bộ điều khiển Công việc yêu cầu máy chủ API tạo hoặc xóa Pod. Các thành phần khác trong control plane hành động dựa trên thông tin mới (có các Pod mới để lên lịch và chạy), và cuối cùng công việc đã hoàn thành.

  Sau khi bạn tạo một Công việc mới, trạng thái mong muốn là Công việc đó đã được hoàn thành. Bộ điều khiển Công việc làm cho trạng thái hiện tại của Công việc đó gần với trạng thái mong muốn hơn: tạo các Pod thực hiện làm việc cho Công việc đó, để Công việc gần hoàn thành hơn.

  Bộ điều khiển cũng cập nhật các đối tượng cấu hình chúng. Ví dụ: sau khi hoàn thành việc làm cho một Công việc, bộ điều khiển Công việc sẽ cập nhật đối tượng Công việc đó để đánh dấu nó `Finished`.

- **Điều khiển trực tiếp**

  Ngược lại với Công việc, một số bộ điều khiển cần thực hiện thay đổi đối với những thứ bên ngoài cụm.

  Ví dụ: nếu bạn sử dụng vòng lặp điều khiển để đảm bảo có đủ các Node trong cụm của bạn, thì bộ điều khiển đó cần thứ gì đó bên ngoài cụm hiện tại để thiết lập các Node mới khi cần.

Các bộ điều khiển tương tác với trạng thái bên ngoài sẽ tìm thấy trạng thái mong muốn của chúng từ máy chủ API, sau đó giao tiếp trực tiếp với hệ thống bên ngoài để đưa trạng thái hiện tại đến gần trạng thái mong muốn hơn.

(Thực sự có một bộ điều khiển chia tỷ lệ các nút trong cụm của bạn theo chiều ngang.)

Điểm quan trọng ở đây là bộ điều khiển thực hiện một số thay đổi để mang lại trạng thái mong muốn, sau đó báo cáo trạng thái hiện tại trở lại máy chủ API của cụm. Các vòng điều khiển khác có thể quan sát dữ liệu được báo cáo đó và thực hiện các hành động của riêng chúng.

Với các cụm Kubernetes, control plane hoạt động gián tiếp với các công cụ quản lý địa chỉ IP, dịch vụ lưu trữ, API của nhà cung cấp đám mây và các dịch vụ khác bằng cách mở rộng Kubernetes để triển khai điều đó.

Mong muốn so với trạng thái hiện tại

Kubernetes có chế độ xem hệ thống dựa trên đám mây và có thể xử lý thay đổi liên tục.

Cụm của bạn có thể thay đổi bất kỳ lúc nào khi công việc diễn ra và các vòng lặp điều khiển sẽ tự động khắc phục lỗi. Điều này có nghĩa là, có khả năng, cụm của bạn không bao giờ đạt đến trạng thái ổn định.

Miễn là các bộ điều khiển cho cụm đang chạy và có thể thực hiện các thay đổi hữu ích, nó không quan trọng nếu trạng thái tổng thể là ổn định hay không.

Thiết kế

Như một nguyên lý trong thiết kế của nó, Kubernetes sử dụng rất nhiều bộ điều khiển mà mỗi bộ điều khiển quản lý một khía cạnh cụ thể của trạng thái cụm. Thông thường nhất, một vòng lặp điều khiển cụ thể (bộ điều khiển) sử dụng một loại tài nguyên làm trạng thái mong muốn của nó và có một loại tài nguyên khác mà nó quản lý để thực hiện trạng thái mong muốn đó. Ví dụ: bộ điều khiển cho Công việc theo dõi các đối tượng Công việc (để khám phá công việc mới) và các đối tượng Pod (để chạy Công việc, sau đó để xem khi nào công việc kết thúc). Trong trường hợp này, một cái gì đó khác tạo ra Công việc, trong khi bộ điều khiển Công việc tạo ra các Pod.

Thật hữu ích khi có các bộ điều khiển đơn giản thay vì một bộ vòng điều khiển nguyên khối được liên kết với nhau. Bộ điều khiển có thể bị lỗi, vì vậy Kubernetes được thiết kế để cho phép điều đó xảy ra.

Ghi chú: Có thể có một số bộ điều khiển tạo hoặc cập nhật cùng một loại đối tượng. Đằng sau hậu trường, bộ điều khiển Kubernetes đảm bảo rằng họ chỉ chú ý đến các tài nguyên được liên kết với tài nguyên kiểm soát của họ.

Ví dụ: bạn có thể có các Deployment và Công việc; cả hai đều tạo ra các Pod. Bộ điều khiển công việc không xóa các Pod mà Deployment của bạn đã tạo, vì có thông tin (nhãn) bộ điều khiển có thể sử dụng để phân biệt các Pod đó.

Các cách chạy bộ điều khiển

Kubernetes đi kèm với một bộ điều khiển tích hợp chạy bên trong kube-controller-manager. Các bộ điều khiển tích hợp này cung cấp các hành vi cốt lõi quan trọng.

Bộ điều khiển Deployment và Bộ điều khiển công việc là những ví dụ về bộ điều khiển là một phần của chính Kubernetes (bộ điều khiển "tích hợp"). Kubernetes cho phép bạn chạy một control plane có khả năng đối phó để nếu bất kỳ bộ điều khiển tích hợp nào bị lỗi, một phần khác của control plane sẽ đảm nhận công việc.

Bạn có thể tìm các bộ điều khiển chạy bên ngoài control plane để mở rộng Kubernetes. Hoặc, nếu muốn, bạn có thể tự viết một bộ điều khiển mới. Bạn có thể chạy bộ điều khiển của riêng mình dưới dạng một bộ Pod hoặc bên ngoài Kubernetes. Những gì phù hợp nhất sẽ phụ thuộc vào chức năng của bộ điều khiển cụ thể đó.

1.3.4 Quản lý controller cloud