Konzepte

- 1: Überblick
  - 1.1: Was ist Kubernetes?
  - 1.2: Kubernetes Komponenten
- 2: Kubernetes Architekur
  - 2.1: Nodes
  - 2.2: Master-Node Kommunikation
  - 2.3: Zugrunde liegende Konzepte des Cloud Controller Manager
- 3: Container
  - 3.1: Images
- 4: Workloads
  - 4.1: Pods
- 5: Dienste, Lastverteilung und Netzwerkfunktionen
- 6: Speicher
- 7: Konfiguration
- 8: Richtlinien
- 9: Cluster Administration
  - 9.1: Proxies in Kubernetes
  - 9.2: Controller Manager Metriken
  - 9.3: Addons Installieren
- 10: Kubernetes erweitern
- 11: Konzept Dokumentations-Vorlage

Im Abschnitt Konzepte erfahren Sie mehr über die Bestandteile des Kubernetes-Systems und die Abstraktionen, die Kubernetes zur Verwaltung Ihres Clusters zur Verfügung stellt. Sie erhalten zudem ein tieferes Verständnis der Funktionsweise von Kubernetes.

Überblick

Um mit Kubernetes zu arbeiten, verwenden Sie *Kubernetes-API-Objekte*, um den *gewünschten Status Ihres Clusters* zu beschreiben: welche Anwendungen oder anderen Workloads Sie ausführen möchten, welche Containerimages sie verwenden, die Anzahl der Replikate, welche Netzwerk- und Festplattenressourcen Sie zur Verfügung stellen möchten, und vieles mehr. Sie legen den gewünschten Status fest, indem Sie Objekte mithilfe der Kubernetes-API erstellen. Dies geschieht normalerweise über die Befehlszeilenschnittstelle `kubectl`. Sie können die Kubernetes-API auch direkt verwenden, um mit dem Cluster zu interagieren und den gewünschten Status festzulegen oder zu ändern.

Sobald Sie den gewünschten Status eingestellt haben, wird das *Kubernetes Control Plane* dafür sorgen, dass der aktuelle Status des Clusters mit dem gewünschten Status übereinstimmt. Zu diesem Zweck führt Kubernetes verschiedene Aufgaben automatisch aus, z. B. das Starten oder Neustarten von Containern, Skalieren der Anzahl der Repliken einer bestimmten Anwendung und vieles mehr. Das Kubernetes Control Plane besteht aus einer Reihe von Prozessen, die in Ihrem Cluster ausgeführt werden:

- Der **Kubernetes Master** bestehet aus drei Prozessen, die auf einem einzelnen Node in Ihrem Cluster ausgeführt werden, der als Master-Node bezeichnet wird. Diese Prozesse sind:kube-apiserver, kube-controller-manager und kube-scheduler.
- Jeder einzelne Node in Ihrem Cluster, welcher nicht der Master ist, führt zwei Prozesse aus:
  - **kubelet**, das mit dem Kubernetes Master kommuniziert.
  - **kube-proxy**, ein Netzwerk-Proxy, der die Netzwerkdienste von Kubernetes auf jedem Node darstellt.

Kubernetes Objects

Kubernetes enthält eine Reihe von Abstraktionen, die den Status Ihres Systems darstellen: im Container eingesetzte Anwendungen und Workloads, die zugehörigen Netzwerk- und Festplattenressourcen sowie weitere Informationen zu den Aufgaben Ihres Clusters. Diese Abstraktionen werden durch Objekte in der Kubernetes-API dargestellt; Lesen Sie Kubernetes Objects Überblick für weitere Details.

Die Basisobjekte von Kubernetes umfassen:

- Pod
- Service
- Volume
- Namespace

Darüber hinaus enthält Kubernetes Abstraktionen auf höherer Ebene, die als Controller bezeichnet werden. Controller bauen auf den Basisobjekten auf und bieten zusätzliche Funktionen und Komfortfunktionen. Sie beinhalten:

- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job

Kubernetes Control Plane

Die verschiedenen Teile der Kubernetes-Steuerungsebene (Control Plane), wie der Kubernetes Master- und der Kubelet-Prozess, bestimmen, wie Kubernetes mit Ihrem Cluster kommuniziert. Das Control Plane verwaltet ein Inventar aller Kubernetes-Objekte im System und führt kontinuierlich Kontrollschleifen aus, um den Status dieser Objekte zu verwalten. Zu jeder Zeit reagieren die Kontrollschleifen des Control Plane auf Änderungen im Cluster und arbeiten daran, dass der tatsächliche Status aller Objekte im System mit dem von Ihnen definierten Status übereinstimmt.

Wenn Sie beispielsweise mit der Kubernetes-API ein Deployment-Objekt erstellen, geben Sie einen neuen gewünschten Status für das System an. Das Kubernetes Control Plane zeichnet die Objekterstellung auf und führt Ihre Anweisungen aus, indem es die erforderlichen Anwendungen startet und Sie für auf den Cluster-Nodes plant - Dadurch wird der tatsächliche Status des Clusters an den gewünschten Status angepasst.

Kubernetes Master

Der Kubernetes-Master ist für Erhalt des gewünschten Status Ihres Clusters verantwortlich. Wenn Sie mit Kubernetes interagieren, beispielsweise mit dem Kommandozeilen-Tool `kubectl`, kommunizieren Sie mit dem Kubernetes-Master Ihres Clusters.

Der Begriff "Master" bezeichnet dabei eine Reihe von Prozessen, die den Clusterstatus verwalten. Normalerweise werden diese Prozesse alle auf einem einzigen Node im Cluster ausgeführt. Dieser Node wird auch als Master bezeichnet. Der Master kann repliziert werden, um die Verfügbarkeit und Redundanz zu erhöhen.

Kubernetes Nodes

Die Nodes in einem Cluster sind die Maschinen (VMs, physische Server usw.), auf denen Ihre Anwendungen und Cloud-Workflows ausgeführt werden. Der Kubernetes-Master steuert jeden Node; Sie werden selten direkt mit Nodes interagieren.

Objekt Metadata

- Anmerkungen

Nächste Schritte

Wenn Sie eine Konzeptseite schreiben möchten, lesen Sie Seitenvorlagen verwenden für Informationen zum Konzeptseitentyp und zur Dokumentations Vorlage.

1 - Überblick

1.1 - Was ist Kubernetes?

Diese Seite ist eine Übersicht über Kubernetes.

Kubernetes ist eine portable, erweiterbare Open-Source-Plattform zur Verwaltung von containerisierten Arbeitslasten und Services, die sowohl die deklarative Konfiguration als auch die Automatisierung erleichtert. Es hat ein großes, schnell wachsendes Ökosystem. Kubernetes Dienstleistungen, Support und Tools sind weit verbreitet.

Google hat das Kubernetes-Projekt 2014 als Open-Source-Projekt zur Verfügung gestellt. Kubernetes baut auf anderthalb Jahrzehnten Erfahrung auf, die Google mit der Ausführung von Produktions-Workloads in großem Maßstab hat, kombiniert mit den besten Ideen und Praktiken der Community.

Warum brauche ich Kubernetes und was kann ich damit tun?

Kubernetes hat eine Reihe von Funktionen. Es kann gesehen werden als:

- eine Containerplattform
- eine Microservices-Plattform
- eine portable Cloud-Plattform und vieles mehr.

Kubernetes bietet eine **containerzentrierte** Managementumgebung. Es koordiniert die Computer-, Netzwerk- und Speicherinfrastruktur im Namen der Benutzer-Workloads. Dies bietet einen Großteil der Einfachheit von Platform as a Service (PaaS) mit der Flexibilität von Infrastructure as a Service (IaaS) und ermöglicht die Portabilität zwischen Infrastrukturanbietern.

Wie ist Kubernetes eine Plattform?

Auch wenn Kubernetes eine Menge Funktionalität bietet, gibt es immer wieder neue Szenarien, die von neuen Funktionen profitieren würden. Anwendungsspezifische Workflows können optimiert werden, um die Entwicklungsgeschwindigkeit zu beschleunigen. Eine zunächst akzeptable Ad-hoc-Orchestrierung erfordert oft eine robuste Automatisierung in großem Maßstab. Aus diesem Grund wurde Kubernetes auch als Plattform für den Aufbau eines Ökosystems von Komponenten und Tools konzipiert, um die Bereitstellung, Skalierung und Verwaltung von Anwendungen zu erleichtern.

Labels ermöglichen es den Benutzern, ihre Ressourcen nach Belieben zu organisieren. Anmerkungen ermöglichen es Benutzern, Ressourcen mit benutzerdefinierten Informationen zu versehen, um ihre Arbeitsabläufe zu vereinfachen und eine einfache Möglichkeit für Managementtools zu bieten, den Status von Kontrollpunkten zu ermitteln.

Darüber hinaus basiert die Kubernetes-Steuerungsebene auf den gleichen APIs, die Entwicklern und Anwendern zur Verfügung stehen. Benutzer können ihre eigenen Controller, wie z.B. Scheduler, mit ihren eigenen APIs schreiben, die von einem universellen Kommandozeilen-Tool angesprochen werden können.

Dieses Design hat es einer Reihe anderer Systeme ermöglicht, auf Kubernetes aufzubauen.

Was Kubernetes nicht ist

Kubernetes ist kein traditionelles, allumfassendes PaaS (Plattform als ein Service) System. Da Kubernetes nicht auf Hardware-, sondern auf Containerebene arbeitet, bietet es einige allgemein anwendbare Funktionen, die PaaS-Angeboten gemeinsam sind, wie Bereitstellung, Skalierung, Lastausgleich, Protokollierung und Überwachung. Kubernetes ist jedoch nicht

monolithisch, und diese Standardlösungen sind optional und modular erweiterbar. Kubernetes liefert die Bausteine für den Aufbau von Entwicklerplattformen, bewahrt aber die Wahlmöglichkeiten und Flexibilität der Benutzer, wo es wichtig ist.

Kubernetes:

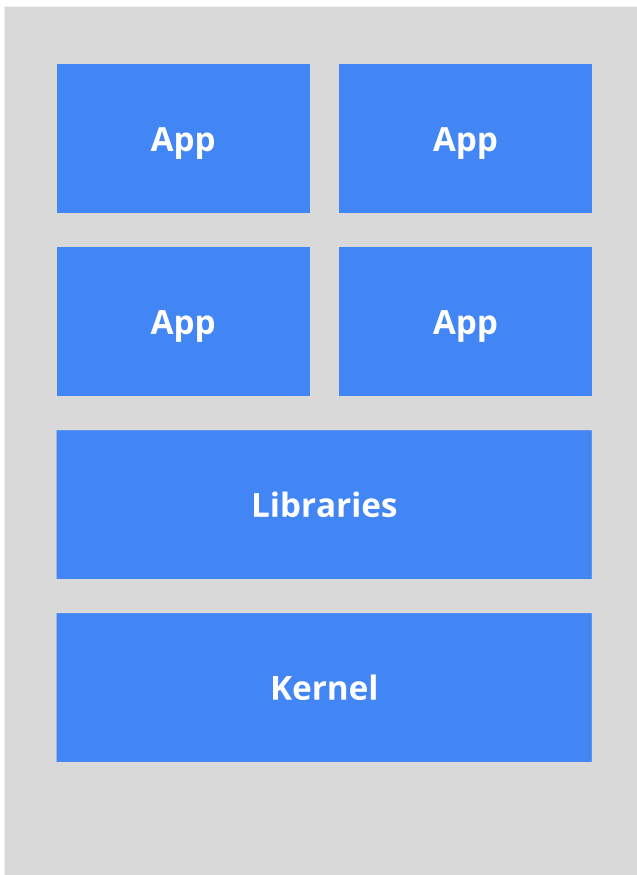
- Schränkt nicht die Art der unterstützten Anwendungen ein. Kubernetes zielt darauf ab, eine extrem große Vielfalt von Workloads zu unterstützen, einschließlich stateless, stateful und datenverarbeitender Workloads. Wenn eine Anwendung in einem Container ausgeführt werden kann, sollte sie auf Kubernetes hervorragend laufen.
- Verteilt keinen Quellcode und entwickelt Ihre Anwendung nicht. Kontinuierliche Integrations-, Liefer- und Bereitstellungs-Workflows (CI/CD) werden durch Unternehmenskulturen und -präferenzen sowie technische Anforderungen bestimmt.
- Bietet keine Dienste auf Anwendungsebene, wie Middleware (z.B. Nachrichtenbusse), Datenverarbeitungs-Frameworks (z.B. Spark), Datenbanken (z.B. mysql), Caches oder Cluster-Speichersysteme (z.B. Ceph) als eingebaute Dienste. Solche Komponenten können auf Kubernetes laufen und/oder von Anwendungen, die auf Kubernetes laufen, über portable Mechanismen wie den Open Service Broker angesprochen werden.
- Bietet keine Konfigurationssprache bzw. kein Konfigurationssystem (z.B. jsonnet). Es bietet eine deklarative API, die von beliebigen Formen deklarativer Spezifikationen angesprochen werden kann.
- Bietet keine umfassenden Systeme zur Maschinenkonfiguration, Wartung, Verwaltung oder Selbstheilung.

Außerdem ist Kubernetes nicht nur ein *Orchestrierungssystem*. Fakt ist, dass es die Notwendigkeit einer Orchestrierung überflüssig macht. Die technische Definition von *Orchestrierung* ist die Ausführung eines definierten Workflows: zuerst A, dann B, dann C. Im Gegensatz dazu besteht Kubernetes aus einer Reihe von unabhängigen, komponierbaren Steuerungsprozessen, die den aktuellen Zustand kontinuierlich in Richtung des bereitgestellten Soll-Zustandes vorantreiben. Es sollte keine Rolle spielen, wie Sie von A nach C kommen. Eine zentrale Steuerung ist ebenfalls nicht erforderlich. Das Ergebnis ist ein System, das einfacher zu bedienen und leistungsfähiger, robuster, widerstandsfähiger und erweiterbar ist.

Warum Container?

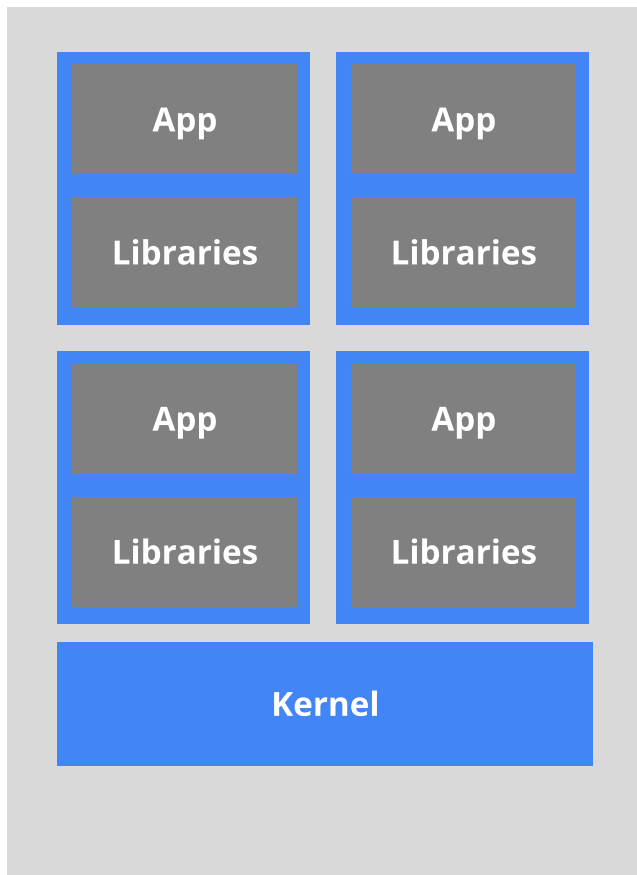
Sie suchen nach Gründen, warum Sie Container verwenden sollten?

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



Small and fast, portable
Uses OS-level virtualization

Der *Altbekannte* Weg zur Bereitstellung von Anwendungen war die Installation der Anwendungen auf einem Host mit dem Betriebssystempaketmanager. Dies hatte den Nachteil, dass die ausführbaren Dateien, Konfigurationen, Bibliotheken und Lebenszyklen der Anwendungen untereinander und mit dem Host-Betriebssystem verwoben waren. Man könnte unveränderliche Virtual-Machine-Images erzeugen, um vorhersehbare Rollouts und Rollbacks zu erreichen, aber VMs sind schwergewichtig und nicht portierbar.

Der *Neue Weg* besteht darin, Container auf Betriebssystemebene und nicht auf Hardware-Virtualisierung bereitzustellen. Diese Container sind voneinander und vom Host isoliert: Sie haben ihre eigenen Dateisysteme, sie können die Prozesse des anderen nicht sehen, und ihr Ressourcenverbrauch kann begrenzt werden. Sie sind einfacher zu erstellen als VMs, und da sie von der zugrunde liegenden Infrastruktur und dem Host-Dateisystem entkoppelt sind, sind sie über Clouds und Betriebssystem-Distributionen hinweg portabel.

Da Container klein und schnell sind, kann in jedes Containerimage eine Anwendung gepackt werden. Diese 1:1-Beziehung zwischen Anwendung und Image ermöglicht es, die Vorteile von Containern voll auszuschöpfen. Mit Containern können unveränderliche Container-Images eher zur Build-/Release-Zeit als zur Deployment-Zeit erstellt werden, da jede Anwendung nicht mit dem Rest des Anwendungsstacks komponiert werden muss und auch nicht mit der Produktionsinfrastrukturumgebung verbunden ist. Die Generierung von Container-Images zum Zeitpunkt der Erstellung bzw. Freigabe ermöglicht es, eine konsistente Umgebung von der Entwicklung bis zur Produktion zu gewährleisten. Ebenso sind Container wesentlich transparenter als VMs, was die Überwachung und Verwaltung erleichtert. Dies gilt insbesondere dann, wenn die Prozesslebenszyklen der Container von der Infrastruktur verwaltet werden und nicht von einem Prozess-Supervisor im Container versteckt werden. Schließlich, mit einer einzigen Anwendung pro Container, wird die Verwaltung der Container gleichbedeutend mit dem Management des Deployments der Anwendung.

Zusammenfassung der Container-Vorteile:

- **Agile Anwendungserstellung und -bereitstellung**: Einfachere und effizientere Erstellung von Container-Images im Vergleich zur Verwendung von VM-Images.
- **Kontinuierliche Entwicklung, Integration und Bereitstellung**: Bietet eine zuverlässige und häufige Erstellung und Bereitstellung von Container-Images mit schnellen und einfachen Rollbacks (aufgrund der Unveränderlichkeit des Images).
- **Dev und Ops Trennung der Bedenken**: Erstellen Sie Anwendungscontainer-Images nicht zum Deployment-, sondern zum Build-Releasezeitpunkt und entkoppeln Sie so Anwendungen von der Infrastruktur.
- **Überwachbarkeit**: Nicht nur Informationen und Metriken auf Betriebssystemebene werden angezeigt, sondern auch der Zustand der Anwendung und andere Signale.
- **Umgebungskontinuität in Entwicklung, Test und Produktion**: Läuft auf einem Laptop genauso wie in der Cloud.
- **Cloud- und OS-Distribution-Portabilität**: Läuft auf Ubuntu, RHEL, CoreOS, On-Prem, Google Kubernetes Engine und überall sonst.
- **Anwendungsorientiertes Management**: Erhöht den Abstraktionsgrad vom Ausführen eines Betriebssystems auf virtueller Hardware bis zum Ausführen einer Anwendung auf einem Betriebssystem unter Verwendung logischer Ressourcen.
- **Locker gekoppelte, verteilte, elastische, freie Microservices**: Anwendungen werden in kleinere, unabhängige Teile zerlegt und können dynamisch bereitgestellt und verwaltet werden -- nicht ein monolithischer Stack, der auf einer großen Single-Purpose-Maschine läuft.
- **Ressourcenisolierung**: Vorhersehbare Anwendungsleistung.
- **Ressourcennutzung**: Hohe Effizienz und Dichte.

Was bedeutet Kubernetes? K8s?

Der Name **Kubernetes** stammt aus dem Griechischen, bedeutet *Steuermann* oder *Pilot*, und ist der Ursprung von *Gouverneur* und cybernetic. *K8s* ist eine Abkürzung, die durch Ersetzen der 8 Buchstaben "ubernete" mit "8" abgeleitet wird.

Nächste Schritte

- Bereit loszulegen?
- Weitere Einzelheiten finden Sie in der Kubernetes Dokumentation.

1.2 - Kubernetes Komponenten

In diesem Dokument werden die verschiedenen binären Komponenten beschrieben, die zur Bereitstellung eines funktionsfähigen Kubernetes-Clusters erforderlich sind.

Master-Komponenten

Master-Komponenten stellen die Steuerungsebene des Clusters bereit. Master-Komponenten treffen globale Entscheidungen über den Cluster (z. B. Zeitplanung) und das Erkennen und Reagieren auf Clusterereignisse (Starten eines neuen Pods, wenn das `replicas` -Feld eines Replikationscontrollers nicht zufriedenstellend ist).

Master-Komponenten können auf jedem Computer im Cluster ausgeführt werden. Der Einfachheit halber starten Setup-Skripts normalerweise alle Master-Komponenten auf demselben Computer, und es werden keine Benutzercontainer auf diesem Computer ausgeführt. Lesen Sie Cluster mit hoher Verfügbarkeit erstellen für ein Beispiel für ein Multi-Master-VM-Setup.

kube-apiserver

Komponente auf dem Master, der die Kubernetes-API verfügbar macht. Es ist das Frontend für die Kubernetes-Steuerebene.

Es ist für die horizontale Skalierung konzipiert, d. H. Es skaliert durch die Bereitstellung von mehr Instanzen. Mehr informationen finden Sie unter Cluster mit hoher Verfügbarkeit erstellen.

etcd

Konsistenter und hochverfügbarer Key-Value Speicher, der als Backupspeicher von Kubernetes für alle Clusterdaten verwendet wird.

Halten Sie immer einen Sicherungsplan für etcds Daten für Ihren Kubernetes-Cluster bereit. Ausführliche Informationen zu etcd finden Sie in der etcd Dokumentation.

kube-scheduler

Komponente auf dem Master, die neu erstellte Pods überwacht, denen kein Node zugewiesen ist. Sie wählt den Node aus, auf dem sie ausgeführt werden sollen.

Zu den Faktoren, die bei Planungsentscheidungen berücksichtigt werden, zählen individuelle und kollektive Ressourcenanforderungen, Hardware- / Software- / Richtlinieneinschränkungen, Affinitäts- und Anti-Affinitätsspezifikationen, Datenlokalität, Interworkload-Interferenz und Deadlines.

kube-controller-manager

Komponente auf dem Master, auf dem controllers ausgeführt werden.

Logisch gesehen ist jeder controller ein separater Prozess, aber zur Vereinfachung der Komplexität werden sie alle zu einer einzigen Binärdatei zusammengefasst und in einem einzigen Prozess ausgeführt.

Diese Controller umfassen:

- Node Controller: Verantwortlich für das Erkennen und Reagieren, wenn Nodes ausfallen.
- Replication Controller: Verantwortlich für die Aufrechterhaltung der korrekten Anzahl von Pods für jedes Replikationscontrollerobjekt im System.
- Endpoints Controller: Füllt das Endpoints-Objekt aus (d.h. verbindet Services & Pods).
- Service Account & Token Controllers: Erstellt Standardkonten und API-Zugriffstoken für neue Namespaces.

cloud-controller-manager

cloud-controller-manager führt Controller aus, die mit den entsprechenden Cloud-Anbietern interagieren. Der cloud-controller-manager ist eine Alpha-Funktion, die in Kubernetes Version 1.6 eingeführt wurde.

cloud-controller-manager führt nur Cloud-Provider-spezifische Controller-Schleifen aus. Sie müssen diese Controller-Schleifen im Cube-Controller-Manager deaktivieren. Sie können die Controller-Schleifen deaktivieren, indem Sie beim Starten des kube-controller-manager das Flag `--cloud-provider` auf `external` setzen.

cloud-controller-manager erlaubt es dem Cloud-Anbieter Code und dem Kubernetes-Code, sich unabhängig voneinander zu entwickeln. In früheren Versionen war der Kerncode von Kubernetes für die Funktionalität von Cloud-Provider-spezifischem Code abhängig. In zukünftigen Versionen sollte der für Cloud-Anbieter spezifische Code vom Cloud-Anbieter selbst verwaltet und mit dem Cloud-Controller-Manager verknüpft werden, während Kubernetes ausgeführt wird.

Die folgenden Controller haben Abhängigkeiten von Cloud-Anbietern:

- Node Controller: Zum Überprüfen, ob ein Node in der Cloud beim Cloud-Anbieter gelöscht wurde, nachdem er nicht mehr reagiert
- Route Controller: Zum Einrichten von Routen in der zugrunde liegenden Cloud-Infrastruktur
- Service Controller: Zum Erstellen, Aktualisieren und Löschen von Lastverteilern von Cloud-Anbietern
- Volume Controller: Zum Erstellen, Verbinden und Bereitstellen von Volumes und zur Interaktion mit dem Cloud-Provider zum Orchestrieren von Volumes

Node-Komponenten

Node Komponenten werden auf jedem Knoten ausgeführt, halten laufende Pods aufrecht und stellen die Kubernetes-Laufzeitumgebung bereit.

kubelet

Ein Agent, der auf jedem Node im Cluster ausgeführt wird. Er stellt sicher, dass Container in einem Pod ausgeführt werden.

Das Kubelet verwendet eine Reihe von PodSpecs, die über verschiedene Mechanismen bereitgestellt werden, und stellt sicher, dass die in diesen PodSpecs beschriebenen Container ordnungsgemäß ausgeführt werden. Das kubelet verwaltet keine Container, die nicht von Kubernetes erstellt wurden.

kube-proxy

kube-proxy ermöglicht die Kubernetes Service-Abstraktion, indem die Netzwerkregeln auf dem Host beibehalten und die Verbindungsweiterleitung durchgeführt wird.

Container Runtime

Die Containerlaufzeit ist die Software, die für das Ausführen von Containern verantwortlich ist. Kubernetes unterstützt mehrere Laufzeiten: Docker, containerd, cri-o, rktlet und jede Implementierung des Kubernetes CRI (Container Runtime Interface).

Addons

Addons sind Pods und Dienste, die Clusterfunktionen implementieren. Die Pods können verwaltet werden durch Deployments, ReplicationControllers, und so wieter. Namespace-Addon-Objekte werden im Namespace `kube-system` erstellt.

Ausgewählte Addons werden unten beschrieben. Eine erweiterte Liste verfügbarer Addons finden Sie unter Addons.

DNS

Während die anderen Addons nicht unbedingt erforderlich sind, sollte cluster DNS in allen Kubernetes-Cluster vorhanden sein, da viele Beispiele davon abhängen.

Cluster-DNS ist neben anderen DNS-Servern in Ihrer Umgebung ein DNS-Server, der DNS-Einträge für Kubernetes-Dienste bereitstellt.

Von Kubernetes gestartete Container schließen diesen DNS-Server automatisch in ihre DNS-Suchen ein.

Web UI (Dashboard)

Dashboard ist eine allgemeine, webbasierte Benutzeroberfläche für Kubernetes-Cluster. Benutzer können damit Anwendungen, die im Cluster ausgeführt werden, sowie den Cluster selbst verwalten und Fehler beheben.

Container Resource Monitoring

Container Resource Monitoring zeichnet generische Zeitreihenmessdaten zu Containern in einer zentralen Datenbank auf und stellt eine Benutzeroberfläche zum Durchsuchen dieser Daten bereit.

Cluster-level Logging

Ein Cluster-level logging Mechanismus ist für das Speichern von Containerprotokollen in einem zentralen Protokollspeicher mit Such- / Browsing-Schnittstelle verantwortlich.

2 - Kubernetes Architekur

2.1 - Nodes

Ein Knoten (Node in Englisch) ist eine Arbeitsmaschine in Kubernetes. Ein Node kann je nach Cluster eine VM oder eine physische Maschine sein. Jeder Node enthält die für den Betrieb von Pods notwendigen Dienste und wird von den Master-Komponenten verwaltet. Die Dienste auf einem Node umfassen die Container Runtime, das Kubelet und den Kube-Proxy. Weitere Informationen finden Sie im Abschnitt Kubernetes Node in der Architekturdesign-Dokumentation.

Node Status

Der Status eines Nodes enthält folgende Informationen:

- Adressen
- Zustand
- Kapazität
- Info

Jeder Abschnitt wird folgend detailliert beschrieben.

Adressen

Die Verwendung dieser Felder hängt von Ihrem Cloud-Anbieter oder der Bare-Metal-Konfiguration ab.

- HostName: Der vom Kernel des Nodes gemeldete Hostname. Kann mit dem kubelet-Parameter `--hostname-override` überschrieben werden.
- ExternalIP: In der Regel die IP-Adresse des Nodes, die extern geroutet werden kann (von außerhalb des Clusters verfügbar).
- InternalIP: In der Regel die IP-Adresse des Nodes, die nur innerhalb des Clusters routbar ist.

Zustand

Das `conditions` Feld beschreibt den Zustand, aller `Running` Nodes.

Node Condition	Beschreibung
`OutOfDisk`	`True` wenn auf dem Node nicht genügend freier Speicherplatz zum Hinzufügen neuer Pods vorhanden ist, andernfalls `False`
`Ready`	`True` wenn der Node in einem guten Zustand und bereit ist Pods aufzunehmen, `False` wenn der Node nicht in einem guten Zustand ist und nicht bereit ist Pods aufzunehmeb, und `Unknown` wenn der Node-Controller seit der letzten `node-monitor-grace-period` nichts von dem Node gehört hat (Die Standardeinstellung beträgt 40 Sekunden)
`MemoryPressure`	`True` wenn der verfügbare Speicher des Nodes niedrig ist; Andernfalls `False`
`PIDPressure`	`True` wenn zu viele Prozesse auf dem Node vorhanden sind; Andernfalls `False`
`DiskPressure`	`True` wenn die Festplattenkapazität niedrig ist. Andernfalls `False`

Node Condition	Beschreibung
`Network Unavailable`	`True` wenn das Netzwerk für den Node nicht korrekt konfiguriert ist, andernfalls `False`

Der Zustand eines Nodes wird als JSON-Objekt dargestellt. Die folgende Antwort beschreibt beispielsweise einen fehlerfreien Node.

```
"conditions": [
  {
    "type": "Ready",
    "status": "True"
  }
]
```

Wenn der Status der `Ready` -Bedingung `Unknown` oder `False` länger als der `pod-eviction-timeout` bleibt, wird ein Parameter an den kube-controller-manager übergeben und alle Pods auf dem Node werden vom Node Controller gelöscht.

Die voreingestellte Zeit vor der Entfernung beträgt **fünf Minuten**. In einigen Fällen, in denen der Node nicht erreichbar ist, kann der Apiserver nicht mit dem Kubelet auf dem Node kommunizieren. Die Entscheidung, die Pods zu löschen, kann dem Kublet erst mitgeteilt werden, wenn die Kommunikation mit dem Apiserver wiederhergestellt ist. In der Zwischenzeit können Pods, deren Löschen geplant ist, weiterhin auf dem unzugänglichen Node laufen.

In Versionen von Kubernetes vor 1.5 würde der Node Controller das Löschen dieser unerreichbaren Pods vom Apiserver erzwingen. In Version 1.5 und höher erzwingt der Node Controller jedoch keine Pod Löschung, bis bestätigt wird, dass sie nicht mehr im Cluster ausgeführt werden. Pods die auf einem unzugänglichen Node laufen sind eventuell in einem einem `Terminating` oder `Unkown` Status. In Fällen, in denen Kubernetes nicht aus der zugrunde liegenden Infrastruktur schließen kann, ob ein Node einen Cluster dauerhaft verlassen hat, muss der Clusteradministrator den Node möglicherweise manuell löschen. Das Löschen des Kubernetes-Nodeobjekts bewirkt, dass alle auf dem Node ausgeführten Pod-Objekte gelöscht und deren Namen freigegeben werden.

In Version 1.12 wurde die Funktion `TaintNodesByCondition` als Beta-Version eingeführt, die es dem Node-Lebenszyklus-Controller ermöglicht, automatisch Markierungen (*taints* in Englisch) zu erstellen, die Bedingungen darstellen.

Ebenso ignoriert der Scheduler die Bedingungen, wenn er einen Node berücksichtigt; stattdessen betrachtet er die Markierungen (taints) des Nodes und die Toleranzen eines Pod.

Anwender können jetzt zwischen dem alten Scheduling-Modell und einem neuen, flexibleren Scheduling-Modell wählen.

Ein Pod, der keine Toleranzen aufweist, wird gemäß dem alten Modell geplant. Aber ein Pod, die die Taints eines bestimmten Node toleriert, kann auf diesem Node geplant werden.

> **Achtung:** Wenn Sie diese Funktion aktivieren, entsteht eine kleine Verzögerung zwischen der Zeit, in der eine Bedingung beobachtet wird, und der Zeit, in der ein Taint entsteht. Diese Verzögerung ist in der Regel kürzer als eine Sekunde, aber sie kann die Anzahl der Pods erhöhen, die erfolgreich geplant, aber vom Kubelet abgelehnt werden.

Kapazität

Beschreibt die auf dem Node verfügbaren Ressourcen: CPU, Speicher und die maximale Anzahl der Pods, die auf dem Node ausgeführt werden können.

Info

Allgemeine Informationen zum Node, z. B. Kernelversion, Kubernetes-Version (kubelet- und kube-Proxy-Version), Docker-Version (falls verwendet), Betriebssystemname. Die Informationen werden von Kubelet vom Node gesammelt.

Management

Im Gegensatz zu Pods und Services, ein Node wird nicht von Kubernetes erstellt: Er wird extern von Cloud-Anbietern wie Google Compute Engine erstellt oder ist in Ihrem Pool physischer oder virtueller Maschinen vorhanden. Wenn Kubernetes also einen Node erstellt, wird ein Objekt erstellt, das den Node darstellt. Nach der Erstellung überprüft Kubernetes, ob der Node gültig ist oder nicht.

Wenn Sie beispielsweise versuchen, einen Node aus folgendem Inhalt zu erstellen:

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "10.240.79.157",
    "labels": {
      "name": "my-first-k8s-node"
    }
  }
}
```

Kubernetes erstellt intern ein Node-Objekt (die Darstellung) und validiert den Node durch Zustandsprüfung basierend auf dem Feld `metadata.name` . Wenn der Node gültig ist, d.h. wenn alle notwendigen Dienste ausgeführt werden, ist er berechtigt, einen Pod auszuführen. Andernfalls wird er für alle Clusteraktivitäten ignoriert, bis er gültig wird.

> **Hinweis:** Kubernetes behält das Objekt für den ungültigen Node und prüft ständig seine Gültigkeit. Sie müssen das Node-Objekt explizit löschen, um diesen Prozess zu stoppen.

Aktuell gibt es drei Komponenten, die mit dem Kubernetes Node-Interface interagieren: Node Controller, Kubelet und Kubectl.

Node Controller

Der Node Controller ist eine Kubernetes-Master-Komponente, die verschiedene Aspekte von Nodes verwaltet.

Der Node Controller hat mehrere Rollen im Leben eines Nodes. Der erste ist die Zuordnung eines CIDR-Blocks zu dem Node, wenn er registriert ist (sofern die CIDR-Zuweisung aktiviert ist).

Die zweite ist, die interne Node-Liste des Node Controllers mit der Liste der verfügbaren Computer des Cloud-Anbieters auf dem neuesten Stand zu halten. Wenn ein Node in einer Cloud-Umgebung ausgeführt wird und sich in einem schlechten Zustand befindet, fragt der Node Controller den Cloud-Anbieter, ob die virtuelle Maschine für diesen Node noch verfügbar ist. Wenn nicht, löscht der Node Controller den Node aus seiner Node-Liste.

Der dritte ist die Überwachung des Zustands der Nodes. Der Node Controller ist dafür verantwortlich, die NodeReady-Bedingung von NodeStatus auf ConditionUnknown zu aktualisieren, wenn ein Node unerreichbar wird (der Node Controller empfängt aus irgendeinem Grund keine Herzschläge mehr, z.B. weil der Node heruntergefahren ist) und später alle Pods aus dem Node zu entfernen (und diese ordnungsgemäss zu beenden), wenn

der Node weiterhin unzugänglich ist. (Die Standard-Timeouts sind 40s, um ConditionUnknown zu melden und 5 Minuten, um mit der Evakuierung der Pods zu beginnen).

Der Node Controller überprüft den Zustand jedes Nodes alle `--node-monitor-period` Sekunden.

In Versionen von Kubernetes vor 1.13 ist NodeStatus der Herzschlag des Nodes. Ab Kubernetes 1.13 wird das Node-Lease-Feature als Alpha-Feature eingeführt (Feature-Gate `NodeLease`, KEP-0009).

Wenn die Node Lease Funktion aktiviert ist, hat jeder Node ein zugeordnetes `Lease` -Objekt im `kube-node-lease` -Namespace, das vom Node regelmäßig erneuert wird. Sowohl NodeStatus als auch Node Lease werden als Herzschläge vom Node aus behandelt. Node Leases werden häufig erneuert, während NodeStatus nur dann vom Node zu Master gemeldet wird, wenn sich etwas ändert oder genügend Zeit vergangen ist (Standard ist 1 Minute, was länger ist als der Standard-Timeout von 40 Sekunden für unerreichbare Nodes). Da Node Leases viel lastärmer sind als NodeStatus, macht diese Funktion den Node Herzschlag sowohl in Bezug auf Skalierbarkeit als auch auf die Leistung deutlich effizienter.

In Kubernetes 1.4 haben wir die Logik der Node-Steuerung aktualisiert, um Fälle besser zu handhaben, in denen eine große Anzahl von Nodes Probleme hat, den Master zu erreichen (z.B. weil der Master Netzwerkprobleme hat). Ab 1.4 betrachtet der Node-Controller den Zustand aller Nodes im Cluster, wenn er eine Entscheidung über die Enterfung eines Pods trifft.

In den meisten Fällen begrenzt der Node-Controller die Entfernungsrate auf `--node-eviction-rate` (Standard 0,1) pro Sekunde, was bedeutet, dass er die Pods nicht von mehr als einem Node pro 10 Sekunden entfernt.

Das Entfernungsverhalten von Nodes ändert sich, wenn ein Node in einer bestimmten Verfügbarkeitszone ungesund wird. Der Node-Controller überprüft gleichzeitig, wie viel Prozent der Nodes in der Zone ungesund sind (NodeReady-Bedingung ist ConditionUnknown oder ConditionFalse). Wenn der Anteil der ungesunden Nodes mindestens `--unhealthy-zone-threshold` (Standard 0,55) beträgt, wird die Entfernungsrate reduziert:

Wenn der Cluster klein ist (d.h. weniger als oder gleich `--large-cluster-size-threshold` Node - Standard 50), werden die Entfernungen gestoppt. Andernfalls wird die Entfernungsrate auf `--secondary-node-eviction-rate` (Standard 0,01) pro Sekunde reduziert.

Der Grund, warum diese Richtlinien pro Verfügbarkeitszone implementiert werden, liegt darin, dass eine Verfügbarkeitszone vom Master unerreichbar werden könnte, während die anderen verbunden bleiben. Wenn Ihr Cluster nicht mehrere Verfügbarkeitszonen von Cloud-Anbietern umfasst, gibt es nur eine Verfügbarkeitszone (den gesamten Cluster).

Ein wichtiger Grund für die Verteilung Ihrer Nodes auf Verfügbarkeitszonen ist, dass die Arbeitsbelastung auf gesunde Zonen verlagert werden kann, wenn eine ganze Zone ausfällt. Wenn also alle Nodes in einer Zone ungesund sind, entfernt Node Controller mit der normalen `--node-eviction-rate` Geschwindigkeit. Der Ausnahmefall ist, wenn alle Zonen völlig ungesund sind (d.h. es gibt keine gesunden Node im Cluster). In diesem Fall geht der Node-Controller davon aus, dass es ein Problem mit der Master-Konnektivität gibt und stoppt alle Entfernungen, bis die Verbindung wiederhergestellt ist.

Ab Kubernetes 1.6 ist der Node-Controller auch für die Entfernung von Pods zuständig, die auf Nodes mit `NoExecute` -Taints laufen, wenn die Pods die Markierungen nicht tolerieren. Zusätzlich ist der NodeController als Alpha-Funktion, die standardmäßig deaktiviert ist, dafür verantwortlich, Taints hinzuzufügen, die Node Probleme, wie `Node unreachable` oder `not ready` entsprechen. Siehe diese Dokumentation für Details über `NoExecute` Taints und die Alpha-Funktion.

Ab Version 1.8 kann der Node-Controller für die Erzeugung von Taints, die Node Bedingungen darstellen, verantwortlich gemacht werden. Dies ist eine Alpha-Funktion der Version 1.8.

Selbstregistrierung von Nodes

Wenn das Kubelet-Flag `--register-node` aktiv ist (Standard), versucht das Kubelet, sich beim API-Server zu registrieren. Dies ist das bevorzugte Muster, das von den meisten Distributionen verwendet wird.

Zur Selbstregistrierung wird das kubelet mit den folgenden Optionen gestartet:

- `--kubeconfig` - Pfad zu Anmeldeinformationen, um sich beim Apiserver zu authentifizieren.
- `--cloud-provider` - Wie man sich mit einem Cloud-Anbieter unterhält, um Metadaten über sich selbst zu lesen.
- `--register-node` - Automatisch beim API-Server registrieren.
- `--register-with-taints` - Registrieren Sie den Node mit der angegebenen Taints-Liste (Kommagetrennt `<key>=<value>:<effect>`). No-op wenn `register-node` false ist.
- `--node-ip` - IP-Adresse des Nodes.
- `--node-labels` - Labels, die bei der Registrierung des Nodes im Cluster hinzugefügt werden sollen (Beachten Sie die Richlinien des NodeRestriction admission plugin in 1.13+).
- `--node-status-update-frequency` - Gibt an, wie oft kubelet den Nodestatus an den Master übermittelt.

Wenn der Node authorization mode und NodeRestriction admission plugin aktiviert sind, dürfen kubelets nur ihre eigene Node-Ressource erstellen / ändern.

Manuelle Nodeverwaltung

Ein Cluster-Administrator kann Nodeobjekte erstellen und ändern.

Wenn der Administrator Nodeobjekte manuell erstellen möchte, setzen Sie das kubelet Flag `--register-node=false` .

Der Administrator kann Node-Ressourcen ändern (unabhängig von der Einstellung von `--register-node`). Zu den Änderungen gehören das Setzen von Labels und das Markieren des Nodes.

Labels auf Nodes können in Verbindung mit node selectors auf Pods verwendet werden, um die Planung zu steuern, z.B. um einen Pod so zu beschränken, dass er nur auf einer Teilmenge der Nodes ausgeführt werden darf.

Das Markieren eines Nodes als nicht geplant, verhindert, dass neue Pods für diesen Node geplant werden. Dies hat jedoch keine Auswirkungen auf vorhandene Pods auf dem Node. Dies ist nützlich als vorbereitender Schritt vor einem Neustart eines Nodes usw. Um beispielsweise einen Node als nicht geplant zu markieren, führen Sie den folgenden Befehl aus:

```
kubectl cordon $NODENAME
```

> **Hinweis:** Pods, die von einem DaemonSet-Controller erstellt wurden, umgehen den Kubernetes-Scheduler und respektieren nicht das *unschedulable* Attribut auf einem Node. Dies setzt voraus, dass Daemons auf dem Computer verbleiben, auch wenn während der Vorbereitung eines Neustarts keine Anwendungen mehr vorhanden sind.

Node Kapazität

Die Kapazität des Nodes (Anzahl der CPU und Speichermenge) ist Teil des Nodeobjekts. Normalerweise registrieren sich Nodes selbst und melden ihre Kapazität beim Erstellen des Nodeobjekts. Sofern Sie Manuelle Nodeverwaltung betreiben, müssen Sie die Node Kapazität setzen, wenn Sie einen Node hinzufügen.

Der Kubernetes-Scheduler stellt sicher, dass für alle Pods auf einem Nodes genügend Ressourcen vorhanden sind. Er prüft, dass die Summe der Requests von Containern auf dem Node nicht größer ist als die Kapazität des Nodes. Er beinhaltet alle Container die vom kubelet gestarted worden, aber keine Container die direkt von der container runtime gestartet wurden, noch jegleiche Prozesse die ausserhalb von Containern laufen.

Wenn Sie Ressourcen explizit für Nicht-Pod-Prozesse reservieren möchten, folgen Sie diesem Lernprogramm um Ressourcen für Systemdaemons zu reservieren.

API-Objekt

Node ist eine Top-Level-Ressource in der Kubernetes-REST-API. Weitere Details zum API-Objekt finden Sie unter: Node API object.

2.2 - Master-Node Kommunikation

Dieses Dokument katalogisiert die Kommunikationspfade zwischen dem Master (eigentlich dem Apiserver) und des Kubernetes-Clusters. Die Absicht besteht darin, Benutzern die Möglichkeit zu geben, ihre Installation so anzupassen, dass die Netzwerkkonfiguration so abgesichert wird, dass der Cluster in einem nicht vertrauenswürdigen Netzwerk (oder mit vollständig öffentlichen IP-Adressen eines Cloud-Providers) ausgeführt werden kann.

Cluster zum Master

Alle Kommunikationspfade vom Cluster zum Master enden beim Apiserver (keine der anderen Master-Komponenten ist dafür ausgelegt, Remote-Services verfügbar zu machen). In einem typischen Setup ist der Apiserver so konfiguriert, dass er Remote-Verbindungen an einem sicheren HTTPS-Port (443) mit einer oder mehreren Formen der Clientauthentifizierung überwacht. Eine oder mehrere Formene von Autorisierung sollte aktiviert sein, insbesondere wenn anonyme Anfragen oder Service Account Tokens aktiviert sind.

Nodes sollten mit dem öffentlichen Stammzertifikat für den Cluster konfiguriert werden, sodass sie eine sichere Verbindung zum Apiserver mit gültigen Client-Anmeldeinformationen herstellen können. Beispielsweise bei einer gewöhnlichen GKE-Konfiguration enstprechen die dem kubelet zur Verfügung gestellten Client-Anmeldeinformationen eines Client-Zertifikats. Lesen Sie über kubelet TLS bootstrapping zur automatisierten Bereitstellung von kubelet-Client-Zertifikaten.

Pods, die eine Verbindung zum Apiserver herstellen möchten, können dies auf sichere Weise tun, indem sie ein Dienstkonto verwenden, sodass Kubernetes das öffentliche Stammzertifikat und ein gültiges Trägertoken automatisch in den Pod einfügt, wenn er instanziiert wird. Der `kubernetes` -Dienst (in allen Namespaces) ist mit einer virtuellen IP-Adresse konfiguriert, die (über den Kube-Proxy) an den HTTPS-Endpunkt auf dem Apiserver umgeleitet wird.

Die Master-Komponenten kommunizieren auch über den sicheren Port mit dem Cluster-Apiserver.

Der Standardbetriebsmodus für Verbindungen vom Cluster (Knoten und Pods, die auf den Knoten ausgeführt werden) zum Master ist daher standardmäßig gesichert und kann über nicht vertrauenswürdige und/oder öffentliche Netzwerke laufen.

Master zum Cluster

Es gibt zwei primäre Kommunikationspfade vom Master (Apiserver) zum Cluster. Der Erste ist vom Apiserver hin zum Kubelet-Prozess, der auf jedem Node im Cluster ausgeführt wird. Der Zweite ist vom Apiserver zu einem beliebigen Node, Pod oder Dienst über die Proxy-Funktionalität des Apiservers.

Apiserver zum kubelet

Die Verbindungen vom Apiserver zum Kubelet werden verwendet für:

- Das Abrufen von Protokollen für Pods.
- Das Verbinden (durch kubectl) mit laufenden Pods.
- Die Bereitstellung der Portweiterleitungsfunktion des kubelet.

Diese Verbindungen enden am HTTPS-Endpunkt des kubelet. Standardmäßig überprüft der Apiserver das Serverzertifikat des Kubelets nicht, was die Verbindung angreifbar für Man-in-the-Middle-Angriffe macht. Die Kommunikation ist daher **unsicher**, wenn die Verbindungen über nicht vertrauenswürdige und/oder öffentliche Netzwerke laufen.

Um diese Verbindung zu überprüfen, verwenden Sie das Flag `--kubelet-certificate-authority` , um dem Apiserver ein Stammzertifikatbündel bereitzustellen, das zur Überprüfung des Server-Zertifikats des kubelets verwendet wird.

Wenn dies nicht möglich ist, verwenden Sie SSH tunneling zwischen dem Apiserver und dem kubelet, falls es erforderlich ist eine Verbindung über ein nicht vertrauenswürdiges oder öffentliches Netz zu vermeiden.

Außerdem sollte Kubelet Authentifizierung und/oder Autorisierung aktiviert sein, um die kubelet-API abzusichern.

Apiserver zu Nodes, Pods und Services

Die Verbindungen vom Apiserver zu einem Node, Pod oder Dienst verwenden standardmäßig einfache HTTP-Verbindungen und werden daher weder authentifiziert noch verschlüsselt. Sie können über eine sichere HTTPS-Verbindung ausgeführt werden, indem dem Node, dem Pod oder dem Servicenamen in der API-URL "https:" vorangestellt wird. Das vom HTTPS-Endpunkt bereitgestellte Zertifikat wird jedoch nicht überprüft, und es werden keine Clientanmeldeinformationen bereitgestellt. Die Verbindung wird zwar verschlüsselt, garantiert jedoch keine Integrität. Diese Verbindungen **sind derzeit nicht sicher** innerhalb von nicht vertrauenswürdigen und/oder öffentlichen Netzen.

SSH Tunnels

Kubernetes unterstützt SSH-Tunnel zum Schutz der Master -> Cluster Kommunikationspfade. In dieser Konfiguration initiiert der Apiserver einen SSH-Tunnel zu jedem Node im Cluster (Verbindung mit dem SSH-Server, der mit Port 22 läuft), und leitet den gesamten Datenverkehr für ein kubelet, einen Node, einen Pod oder einen Dienst durch den Tunnel. Dieser Tunnel stellt sicher, dass der Datenverkehr nicht außerhalb des Netzwerks sichtbar ist, in dem die Knoten ausgeführt werden.

SSH-Tunnel werden zur Zeit nicht unterstützt. Sie sollten also nicht verwendet werden, sei denn, man weiß, was man tut. Ein Ersatz für diesen Kommunikationskanal wird entwickelt.

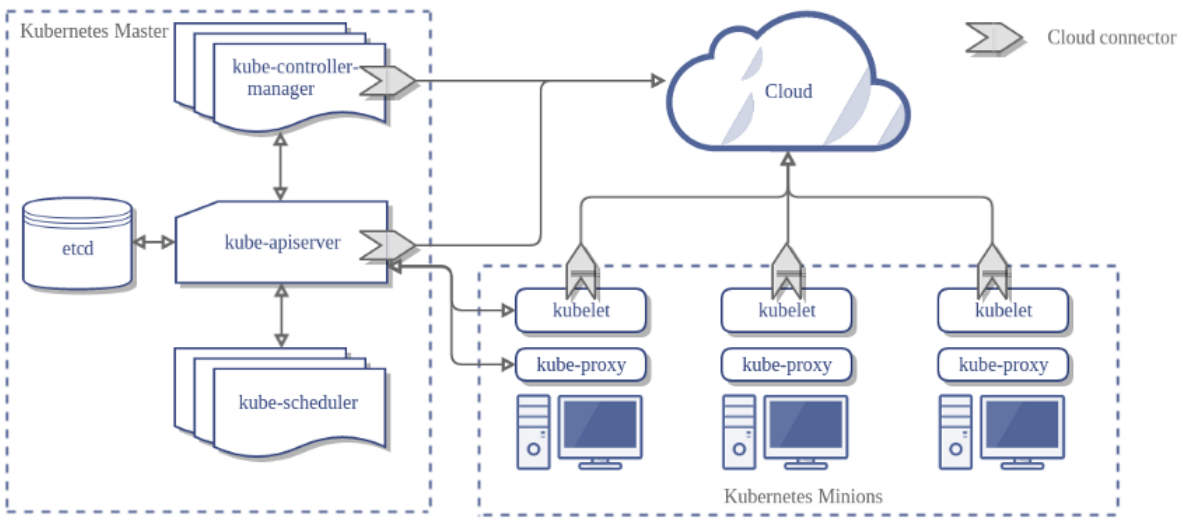
2.3 - Zugrunde liegende Konzepte des Cloud Controller Manager

Das Konzept des Cloud Controller Managers (CCM) (nicht zu verwechseln mit der Binärdatei) wurde ursprünglich entwickelt, um Cloud-spezifischen Anbieter Code und den Kubernetes Kern unabhängig voneinander entwickeln zu können. Der Cloud Controller Manager läuft zusammen mit anderen Master Komponenten wie dem Kubernetes Controller Manager, dem API-Server und dem Scheduler auf dem Host. Es kann auch als Kubernetes Addon gestartet werden, in diesem Fall läuft er auf Kubernetes.

Das Design des Cloud Controller Managers basiert auf einem Plugin Mechanismus, der es neuen Cloud Anbietern ermöglicht, sich mit Kubernetes einfach über Plugins zu integrieren. Es gibt Pläne für die Einbindung neuer Cloud Anbieter auf Kubernetes und für die Migration von Cloud Anbietern vom alten Modell auf das neue CCM-Modell.

Dieses Dokument beschreibt die Konzepte hinter dem Cloud Controller Manager und gibt Details zu den damit verbundenen Funktionen.

Die Architektur eines Kubernetes Clusters ohne den Cloud Controller Manager sieht wie folgt aus:

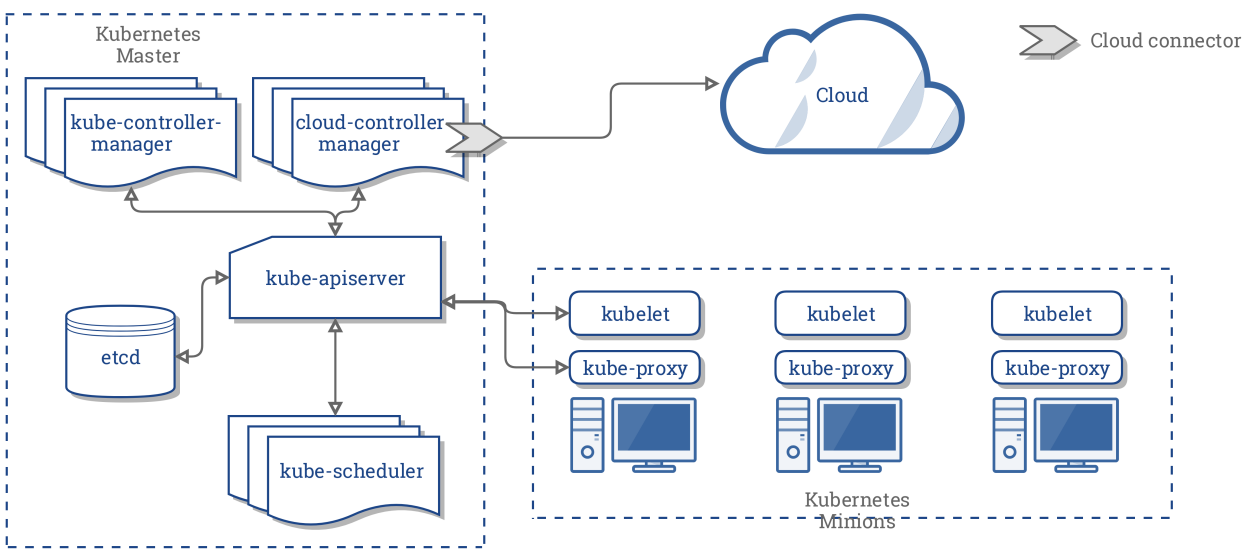


Design

Im vorhergehenden Diagramm sind Kubernetes und der Cloud-Provider über mehrere verschiedene Komponenten integriert:

- Kubelet
- Kubernetes Controller Manager
- Kubernetes API Server

CCM konsolidiert die gesamte Abhängigkeit der Cloud Logik von den drei vorhergehenden Komponenten zu einem einzigen Integrationspunkt mit der Cloud. So sieht die neue Architektur mit dem CCM aus:



Komponenten des CCM

Der CCM löst einen Teil der Funktionalität des Kubernetes Controller Managers (KCM) ab und führt ihn als separaten Prozess aus. Konkret trennt es die Cloud abhängigen Controller im KCM. Der KCM verfügt über die folgenden Cloud abhängigen Steuerungsschleifen:

- Node Controller
- Volume Controller
- Route Controller
- Service Controller

In der Version 1.9 führt der CCM die folgenden Controller aus der vorhergehenden Liste aus:

- Node Controller
- Route Controller
- Service Controller

> **Hinweis:** Der Volume Controller wurde bewusst nicht als Teil des CCM gewählt. Aufgrund der Komplexität und der bestehenden Bemühungen, herstellerspezifische Volume Logik zu abstrahieren, wurde entschieden, dass der Volume Controller nicht zum CCM verschoben wird.

Der ursprüngliche Plan, Volumes mit CCM zu integrieren, sah die Verwendung von Flex-Volumes vor welche austauschbare Volumes unterstützt. Allerdings ist eine konkurrierende Initiative namens CSI geplant, um Flex zu ersetzen.

In Anbetracht dieser Dynamik haben wir uns entschieden, eine Zwischenstopp durchzuführen um die Unterschiede zu beobachten , bis das CSI bereit ist.

Funktionen des CCM

Der CCM erbt seine Funktionen von Komponenten des Kubernetes, die von einem Cloud Provider abhängig sind. Dieser Abschnitt ist auf der Grundlage dieser Komponenten strukturiert.

1. Kubernetes Controller Manager

Die meisten Funktionen des CCM stammen aus dem KCM. Wie im vorherigen Abschnitt erwähnt, führt das CCM die folgenden Steuerschleifen durch:

- Node Controller
- Route Controller
- Service Controller

Node Controller

Der Node Controller ist für die Initialisierung eines Knotens verantwortlich, indem er Informationen über die im Cluster laufenden Knoten vom Cloud Provider erhält. Der Node Controller führt die folgenden Funktionen aus:

1. Initialisierung eines Knoten mit Cloud-spezifischen Zonen-/Regionen Labels.
2. Initialisieren von Knoten mit Cloud-spezifischen Instanzdetails, z.B. Typ und Größe.
3. Ermitteln der Netzwerkadressen und des Hostnamen des Knotens.
4. Falls ein Knoten nicht mehr reagiert, überprüft der Controller die Cloud, um festzustellen, ob der Knoten aus der Cloud gelöscht wurde. Wenn der Knoten aus der Cloud gelöscht wurde, löscht der Controller das Kubernetes Node Objekt.

Route Controller

Der Route Controller ist dafür verantwortlich, Routen in der Cloud so zu konfigurieren, dass Container auf verschiedenen Knoten im Kubernetes Cluster miteinander kommunizieren können. Der Route Controller ist nur auf einem Google Compute Engine Cluster anwendbar.

Service Controller

Der Service Controller ist verantwortlich für das Abhören von Ereignissen zum Erstellen, Aktualisieren und Löschen von Diensten. Basierend auf dem aktuellen Stand der Services in Kubernetes konfiguriert es Cloud Load Balancer (wie ELB, Google LB oder Oracle Cloud Infrastructure LB), um den Zustand der Services in Kubernetes abzubilden. Darüber hinaus wird sichergestellt, dass die Service Backends für Cloud Loadbalancer auf dem neuesten Stand sind.

2. Kubelet

Der Node Controller enthält die Cloud-abhängige Funktionalität des Kubelets. Vor der Einführung des CCM war das Kubelet für die Initialisierung eines Knotens mit cloudspezifischen Details wie IP-Adressen, Regions-/Zonenbezeichnungen und Instanztypinformationen verantwortlich. Mit der Einführung des CCM wurde diese Initialisierungsoperation aus dem Kubelet in das CCM verschoben.

In diesem neuen Modell initialisiert das Kubelet einen Knoten ohne cloudspezifische Informationen. Es fügt jedoch dem neu angelegten Knoten einen Taint hinzu, der den Knoten nicht verplanbar macht, bis der CCM den Knoten mit cloudspezifischen Informationen initialisiert. Dann wird der Taint entfernt.

Plugin Mechanismus

Der Cloud Controller Manager verwendet die Go Schnittstellen, um Implementierungen aus jeder Cloud einzubinden. Konkret verwendet dieser das CloudProvider Interface, das hier definiert ist.

Die Implementierung der vier oben genannten geteiltent Controllern und einigen Scaffolding sowie die gemeinsame CloudProvider Schnittstelle bleiben im Kubernetes Kern. Cloud Provider spezifische Implementierungen werden außerhalb des Kerns aufgebaut und implementieren im Kern definierte Schnittstellen.

Weitere Informationen zur Entwicklung von Plugins findest du im Bereich Entwickeln von Cloud Controller Manager.

Autorisierung

Dieser Abschnitt beschreibt den Zugriff, den der CCM für die Ausführung seiner Operationen auf verschiedene API Objekte benötigt.

Node Controller

Der Node Controller funktioniert nur mit Node Objekten. Es benötigt vollen Zugang zu get, list, create, update, patch, watch, und delete Node Objekten.

v1/Node:

- Get
- List
- Create
- Update
- Patch
- Watch
- Delete

Route Controller

Der Route Controller horcht auf die Erstellung von Node Objekten und konfiguriert die Routen entsprechend. Es erfordert get Zugriff auf die Node Objekte.

v1/Node:

- Get

Service Controller

Der Service Controller hört auf die Service Objekt Events create, update und delete und konfiguriert dann die Endpunkte für diese Services entsprechend.

Um auf Services zuzugreifen, benötigt man list und watch Berechtigung. Um die Services zu aktualisieren, sind patch und update Zugriffsrechte erforderlich.

Um die Endpunkte für die Dienste einzurichten, benötigt der Controller Zugriff auf create, list, get, watch und update.

v1/Service:

- List
- Get
- Watch
- Patch
- Update

Sonstiges

Die Implementierung des Kerns des CCM erfordert den Zugriff auf die Erstellung von Ereignissen und zur Gewährleistung eines sicheren Betriebs den Zugriff auf die Erstellung von ServiceAccounts.

v1/Event:

- Create
- Patch
- Update

v1/ServiceAccount:

- Create

Die RBAC ClusterRole für CCM sieht wie folgt aus:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cloud-controller-manager
rules:
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
  - update
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - '*'
- apiGroups:
```

```
    - ""
    resources:
    - nodes/status
    verbs:
    - patch
  - apiGroups:
    - ""
    resources:
    - services
    verbs:
    - list
    - patch
    - update
    - watch
  - apiGroups:
    - ""
    resources:
    - serviceaccounts
    verbs:
    - create
  - apiGroups:
    - ""
    resources:
    - persistentvolumes
    verbs:
    - get
    - list
    - update
    - watch
  - apiGroups:
    - ""
    resources:
    - endpoints
    verbs:
    - create
    - get
    - list
    - watch
    - update
```

Anbieter Implementierung

Die folgenden Cloud Anbieter haben CCMs implementiert:

- Digital Ocean
- Oracle
- Azure
- GCP
- AWS
- BaiduCloud
- Linode

Cluster Administration

Eine vollständige Anleitung zur Konfiguration und zum Betrieb des CCM findest du hier.

3 - Container

3.1 - Images

Sie erstellen ihr Docker Image und laden es in eine Registry hoch, bevor es in einem Kubernetes Pod referenziert werden kann.

Die `image` Eigenschaft eines Containers unterstüzt die gleiche Syntax wie die des `docker` Kommandos, inklusive privater Registries und Tags.

Aktualisieren von Images

Die Standardregel für das Herunterladen von Images ist `IfNotPresent` , dies führt dazu, dass das Kubelet Images überspringt, die bereits auf einem Node vorliegen. Wenn sie stattdessen möchten, dass ein Image immer forciert heruntergeladen wird, können sie folgendes tun:

- Die `imagePullPolicy` des Containers auf `Always` setzen.
- Die `imagePullPolicy` auslassen und `:latest` als Image Tag nutzen.
- Die `imagePullPolicy` und den Tag des Images auslassen.
- Den AlwaysPullImages Admission Controller aktivieren.

Beachten Sie, dass Sie die Nutzung des `:latest` Tags vermeiden sollten. Für weitere Informationen siehe: Best Practices for Configuration.

Multi-Architektur Images mit Manifesten bauen

Das Docker Kommandozeilenwerkzeug unterstützt jetzt das Kommando `docker manifest` mit den Subkommandos `create` , `annotate` and `push` . Diese Kommandos können dazu genutzt werden Manifeste zu bauen und diese hochzuladen.

Weitere Informationen finden sie in der Docker Dokumentation: https://docs.docker.com/edge/engine/reference/commandline/manifest/

Hier einige Beispiele wie wir dies in unserem Build - Prozess nutzen: https://cs.k8s.io/?q=docker%20manifest%20(create%7Cpush%7Cannotate)&i=nope&files=&repos=

Diese Kommandos basieren rein auf dem Docker Kommandozeileninterface und werden auch damit ausgeführt. Sie sollten entweder die Datei `$HOME/.docker/config.json` bearbeiten und den `experimental` Schlüssel auf `enabled` setzen, oder einfach die Umgebungsvariable `DOCKER_CLI_EXPERIMENTAL` auf `enabled` setzen, wenn Sie das Docker Kommandozeileninterface aufrufen.

> **Hinweis:** Nutzen die bitte Docker *18.06 oder neuer*, ältere Versionen haben entweder Bugs oder unterstützen die experimentelle Kommandozeilenoption nicht. Beispiel: https://github.com/docker/cli/issues/1135 verursacht Probleme unter containerd.

Wenn mit alten Manifesten Probleme auftreten, können sie die alten Manifeste in `$HOME/.docker/manifests` entfernen, um von vorne zu beginnen.

Für Kubernetes selbst nutzen wir typischerweise Images mit dem Suffix `-$(ARCH)` . Um die Abwärtskompatibilität zu erhalten, bitten wir Sie, die älteren Images mit diesen Suffixen zu generieren. Die Idee dahinter ist z.B., das `pause` image zu generieren, welches das Manifest für alle Architekturen hat, `pause-amd64` wäre dann abwärtskompatibel zu älteren Konfigurationen, oder YAML - Dateien, die ein Image mit Suffixen hart kodiert enthalten.

Nutzung einer privaten Registry

Private Registries könnten Schlüssel erfordern um Images von ihnen herunterzuladen. Authentifizierungsdaten können auf verschiedene Weisen hinterlegt werden:

- Bei der Google Container Registry
  - Je Cluster
  - Automatisch in der Google Compute Engine oder Google Kubernetes Engine
  - Allen Pods erlauben von der privaten Registry des Projektes lesen zu können
- Bei der Amazon Elastic Container Registry (ECR)
  - IAM Rollen und Richtlinien nutzen um den Zugriff auf ECR Repositories zu kontrollieren
  - Automatisch ECR Authentifizierungsdaten aktualisieren
- Bei der Oracle Cloud Infrastructure Registry (OCIR)
  - IAM Rollen und Richtlinien nutzen um den Zugriff auf OCIR Repositories zu kontrollieren
- Bei der Azure Container Registry (ACR)
- Bei der IBM Cloud Container Registry
- Nodes konfigurieren sich bei einer privaten Registry authentifizieren zu können - Allen Pods erlauben von jeder konfigurierten privaten Registry lesen zu können - Setzt die Konfiguration der Nodes durch einen Cluster - Aministrator voraus
- Im Voraus heruntergeladene Images
  - Alle Pods können jedes gecachte Image auf einem Node nutzen
  - Setzt root - Zugriff auf allen Nodes zum Einrichten voraus
- Spezifizieren eines ImagePullSecrets auf einem Pod
  - Nur Pods, die eigene Secret tragen, haben Zugriff auf eine private Registry

Jede Option wird im Folgenden im Detail beschrieben

Bei Nutzung der Google Container Registry

Kubernetes hat eine native Unterstützung für die Google Container Registry (GCR) wenn es auf der Google Compute Engine (GCE) läuft. Wenn Sie ihren Cluster auf GCE oder der Google Kubernetes Engine betreiben, genügt es, einfach den vollen Image Namen zu nutzen (z.B. gcr.io/my_project/image:tag).

Alle Pods in einem Cluster haben dann Lesezugriff auf Images in dieser Registry.

Das Kubelet authentifiziert sich bei GCR mit Nutzung des Google service Kontos der jeweiligen Instanz. Das Google Servicekonto der Instanz hat einen `https://www.googleapis.com/auth/devstorage.read_only` , so kann es vom GCR des Projektes herunter, aber nicht hochladen.

Bei Nutzung der Amazon Elastic Container Registry

Kubernetes bietet native Unterstützung für die Amazon Elastic Container Registry, wenn Knoten AWS EC2 Instanzen sind.

Es muss einfach nur der komplette Image Name (z.B. `ACCOUNT.dkr.ecr.REGION.amazonaws.com/imagename:tag`) in der Pod - Definition genutzt werden.

Alle Benutzer eines Clusters, die Pods erstellen dürfen, können dann jedes der Images in der ECR Registry zum Ausführen von Pods nutzen.

Das Kubelet wird periodisch ECR Zugriffsdaten herunterladen und auffrischen, es benötigt hierfür die folgenden Berechtigungen:

- `ecr:GetAuthorizationToken`
- `ecr:BatchCheckLayerAvailability`
- `ecr:GetDownloadUrlForLayer`
- `ecr:GetRepositoryPolicy`

- `ecr:DescribeRepositories`
- `ecr:ListImages`
- `ecr:BatchGetImage`

Voraussetzungen:

- Sie müssen Kubelet in der Version `v1.2.0` nutzen. (Führen sie z.B. (e.g. run `/usr/bin/kubelet --version=true` aus um die Version zu prüfen)
- Sie benötigen Version `v1.3.0` oder neuer wenn ihre Knoten in einer A - Region sind und sich ihre Registry in einer anderen B - Region befindet.
- ECR muss in ihrer Region angeboten werden

Fehlerbehebung:

- Die oben genannten Voraussetzungen müssen erfüllt sein
- Laden sie die $REGION (z.B. `us-west-2`) Zugriffsdaten auf ihren Arbeitsrechner. Verbinden sie sich per SSH auf den Host und nutzen die Docker mit diesen Daten. Funktioniert es?
- Prüfen sie ob das Kubelet it dem Parameter `--cloud-provider=aws` läuft.
- Prüfen sie die Logs des Kubelets (z.B. mit `journalctl -u kubelet`) auf Zeilen wie:
  - `plugins.go:56] Registering credential provider: aws-ecr-key`
  - `provider.go:91] Refreshing cache for provider: *aws_credentials.ecrProvider`

Bei Nutzung der Azure Container Registry (ACR)

Bei Nutzung der Azure Container Registry können sie sich entweder als ein administrativer Nutzer, oder als ein Service Principal authentifizieren. In jedem Fall wird die Authentifizierung über die Standard - Docker Authentifizierung ausgeführt. Diese Anleitung bezieht sich auf das azure-cli Kommandozeilenwerkzeug.

Sie müssen zunächst eine Registry und Authentifizierungsdaten erstellen, eine komplette Dokumentation dazu finden sie hier: Azure container registry documentation.

Sobald sie ihre Container Registry erstellt haben, nutzen sie die folgenden Authentifizierungsdaten:

- `DOCKER_USER` : Service Principal oder Administratorbenutzername
- `DOCKER_PASSWORD` : Service Principal Password oder Administratorpasswort
- `DOCKER_REGISTRY_SERVER` : `${some-registry-name}.azurecr.io`
- `DOCKER_EMAIL` : `${some-email-address}`

Wenn sie diese Variablen befüllt haben, können sie: Kubernetes konfigurieren und damit einen Pod deployen.

Bei Nutzung der IBM Cloud Container Registry

Die IBM Cloud Container Registry bietet eine mandantenfähige Private Image Registry, die Sie nutzen können, um ihre Docker Images sicher zu speichern und zu teilen. Standardmäßig werden Images in ihrer Private Registry vom integrierten Schwachstellenscaner durchsucht, um Sicherheitsprobleme und potentielle Schwachstellen zu finden. Benutzer können ihren IBM Cloud Account nutzen, um Zugang zu ihren Images zu erhalten, oder um einen Token zu generieren, der Zugriff auf die Registry Namespaces erlaubt.

Um das IBM Cloud Container Registry Kommandozeilenwerkzeug zu installieren und einen Namespace für ihre Images zu erstellen, folgen sie dieser Dokumentation Getting started with IBM Cloud Container Registry.

Sie können die IBM Cloud Container Registry nutzen, um Container aus IBM Cloud public images und ihren eigenen Images in den `default` Namespace ihres IBM Cloud Kubernetes Service Clusters zu deployen. Um einen Container in einen anderen Namespace, oder um ein

Image aus einer anderen IBM Cloud Container Registry Region oder einem IBM Cloud account zu deployen, erstellen sie ein Kubernetes `imagePullSecret` . Weitere Informationen finden sie unter: [Building containers from images](#).

Knoten für die Nutzung einer Private Registry konfigurieren

> **Hinweis:** Wenn sie Google Kubernetes Engine verwenden, gibt es schon eine `.dockercfg` auf jedem Knoten, die Zugriffsdaten für ihre Google Container Registry beinhaltet. Dann kann die folgende Vorgehensweise nicht angewendet werden.

> **Hinweis:** Wenn sie AWS EC2 verwenden und die EC2 Container Registry (ECR) nutzen, wird das Kubelet auf jedem Knoten die ECR Zugriffsdaten verwalten und aktualisieren. Dann kann die folgende Vorgehensweise nicht angewendet werden.

> **Hinweis:** Diese Vorgehensweise ist anwendbar, wenn sie ihre Knoten-Konfiguration ändern können; Sie wird nicht zuverlässig auf GCE oder einem anderen Cloud - Provider funktionieren, der automatisch Knoten ersetzt.

> **Hinweis:** Kubernetes unterstützt zurzeit nur die `auths` und `HttpHeaders` Abschnitte der Dockerkonfiguration. Das bedeutet, dass die Hilfswerkzeuge (`credHelpers` ooderr `credsStore`) nicht unterstützt werden.

Docker speichert Schlüssel für eigene Registries entweder unter `$HOME/.dockercfg` oder `$HOME/.docker/config.json` . Wenn sie die gleiche Datei in einen der unten aufgeführten Suchpfade speichern, wird Kubelet sie als Hilfswerkzeug für Zugriffsdaten beim Beziehen von Images nutzen.

- `{--root-dir:-/var/lib/kubelet}/config.json`
- `{cwd of kubelet}/config.json`
- `${HOME}/.docker/config.json`
- `/.docker/config.json`
- `{--root-dir:-/var/lib/kubelet}/.dockercfg`
- `{cwd of kubelet}/.dockercfg`
- `${HOME}/.dockercfg`
- `/.dockercfg`

> **Hinweis:** Eventuell müssen sie `HOME=/root` in ihrer Umgebungsvariablendatei setzen.

Dies sind die empfohlenen Schritte, um ihre Knoten für eine Nutzung einer eigenen Registry zu konfigurieren. In diesem Beispiel führen sie folgende Schritte auf ihrem Desktop/Laptop aus:

1. Führen sie `docker login [server]` für jeden Satz ihrer Zugriffsdaten aus. Dies aktualisiert `$HOME/.docker/config.json` .
2. Prüfen Sie `$HOME/.docker/config.json` in einem Editor darauf, ob dort nur Zugriffsdaten enthalten sind, die Sie nutzen möchten.
3. Erhalten sie eine Liste ihrer Knoten:
   - Wenn sie die Namen benötigen: `nodes=$(kubectl get nodes -o jsonpath='{range.items[*].metadata}{.name} {end}')`
   - Wenn sie die IP - Adressen benötigen: `nodes=$(kubectl get nodes -o jsonpath='{range .items[*].status.addresses[?(@.type=="ExternalIP")]}{.address} {end}')`
4. Kopieren sie ihre lokale `.docker/config.json` in einen der oben genannten Suchpfade.

- Zum Beispiel: `for n in $nodes; do scp ~/.docker/config.json root@$n:/var/lib/kubelet/config.json; done`

Prüfen durch das Erstellen eines Pods, der ein eigenes Image nutzt, z.B.:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: private-image-test-1
spec:
  containers:
    - name: uses-private-image
      image: $PRIVATE_IMAGE_NAME
      imagePullPolicy: Always
      command: [ "echo", "SUCCESS" ]
EOF
pod/private-image-test-1 created
```

Wenn alles funktioniert, sollten sie nach einigen Momenten folgendes sehen:

```
kubectl logs private-image-test-1
SUCCESS
```

Wenn es nicht funktioniert, sehen Sie:

```
kubectl describe pods/private-image-test-1 | grep "Failed"
  Fri, 26 Jun 2015 15:36:13 -0700    Fri, 26 Jun 2015 15:39:13 -0700    19    {ku
```

Sie müssen sich darum kümmern, dass alle Knoten im Cluster die gleiche `.docker/config.json` haben, andernfalls werden Pods auf einigen Knoten starten, auf anderen jedoch nicht. Wenn sie zum Beispiel Knoten automatisch skalieren lassen, sollte dann jedes Instanztemplate die `.docker/config.json` beinhalten, oder ein Laufwerk einhängen, das diese beinhaltet.

Alle Pods haben Lesezugriff auf jedes Image in ihrer eigenen Registry, sobald die Registry - Schlüssel zur `.docker/config.json` hinzugefügt wurden.

Im Voraus heruntergeladene Images

> **Hinweis:** Wenn sie Google Kubernetes Engine verwenden, gibt es schon eine `.dockercfg` auf jedem Knoten die Zugriffsdaten für ihre Google Container Registry beinhaltet. Dann kann die folgende Vorgehensweise nicht angewendet werden.

> **Hinweis:** Diese Vorgehensweise ist anwendbar, wenn sie ihre Knoten-Konfiguration ändern können; Sie wird nicht zuverlässig auf GCE oder einem anderen Cloud - Provider funktionieren, der automatisch Knoten ersetzt.

Standardmäßig wird das Kubelet versuchen, jedes Image von der spezifizierten Registry herunterzuladen. Falls jedoch die `imagePullPolicy` Eigenschaft der Containers auf `IfNotPresent` oder `Never` gesetzt wurde, wird ein lokales Image genutzt (präferiert oder exklusiv, jenachdem).

Wenn Sie sich auf im Voraus heruntergeladene Images als Ersatz für eine Registry - Authentifizierung verlassen möchten, müssen sie sicherstellen, dass alle Knoten die gleichen, im Voraus heruntergeladenen Images aufweisen.

Diese Medthode kann dazu genutzt werden, bestimmte Images aus Geschwindigkeitsgründen im Voraus zu laden, oder als Alternative zur Authentifizierung an einer eigenen Registry zu nutzen.

Alle Pods haben Leserechte auf alle im Voraus geladenen Images.

Spezifizieren eines ImagePullSecrets für einen Pod

> **Hinweis:** Diese Vorgehensweise ist aktuell die empfohlene Vorgehensweise für Google Kubernetes Engine, GCE, und jeden Cloud - Provider bei dem die Knotenerstelltung automatisiert ist.

Kubernetes unterstützt die Spezifikation von Registrierungsschlüsseln für einen Pod.

Erstellung eines Secrets mit einer Docker Konfiguration

Führen sie folgenden Befehl mit Ersetzung der groß geschriebenen Werte aus:

```
kubectl create secret docker-registry <name> --docker-server=DOCKER_REGISTRY_SERV
```

Wenn Sie bereits eine Datei mit Docker-Zugriffsdaten haben, können Sie die Zugriffsdaten als ein Kubernetes Secret importieren: Create a Secret based on existing Docker credentials beschreibt die Erstellung. Dies ist insbesondere dann sinnvoll, wenn sie mehrere eigene Container Registries nutzen, da `kubectl create secret docker-registry` ein Secret erstellt, das nur mit einer einzelnen eigenen Registry funktioniert.

> **Hinweis:** Pods können nur eigene Image Pull Secret in ihrem eigenen Namespace referenzieren, somit muss dieser Prozess jedes mal einzeln für jeden Namespace angewendet werden.

Referenzierung eines imagePullSecrets bei einem Pod

Nun können Sie Pods erstellen, die dieses Secret referenzieren, indem Sie einen Aschnitt `imagePullSecrets` zu ihrer Pod - Definition hinzufügen.

```
cat <<EOF > pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
  imagePullSecrets:
    - name: myregistrykey
EOF

cat <<EOF >> ./kustomization.yaml
resources:
- pod.yaml
EOF
```

Dies muss für jeden Pod getan werden, der eine eigene Registry nutzt.

Die Erstellung dieser Sektion kann jedoch automatisiert werden, indem man imagePullSecrets einer serviceAccount Ressource hinzufügt. Add ImagePullSecrets to a Service Account bietet detaillierte Anweisungen hierzu.

Sie können dies in Verbindung mit einer auf jedem Knoten genutzten `.docker/config.json` benutzen, die Zugriffsdaten werden dann zusammengeführt. Dieser Vorgehensweise wird in der Google Kubernetes Engine funktionieren.

Anwendungsfälle

Es gibt eine Anzahl an Lösungen um eigene Registries zu konfigurieren, hier sind einige Anwendungsfälle und empfohlene Lösungen.

1. Cluster die nur nicht-proprietäre Images (z.B. open-source) ausführen. Images müssen nicht versteckt werden.
   - Nutzung von öffentlichen Images auf Docker Hub.
     - Keine Konfiguration notwendig.
     - Auf GCE/Google Kubernetes Engine, wird automatisch ein lokaler Spiegel für eine verbesserte Geschwindigkeit und Verfügbarkeit genutzt.
2. Cluster die einige proprietäre Images ausführen die für Außenstehende nicht sichtbar sein dürfen, aber für alle Cluster - Benutzer sichtbar sein sollen.
   - Nutzung einer gehosteten privaten Registry Docker registry.
     - Kann auf Docker Hub, oder woanders gehostet werden.
     - Manuelle Konfiguration der .docker/config.json auf jedem Knoten, wie oben beschrieben.
   - Der Betrieb einer internen privaten Registry hinter ihrer Firewall mit offenen Leseberechtigungen.
     - Keine Kubernetes - Konfiguration notwendig
   - Wenn GCE/Google Kubernetes Engine genutzt wird, nutzen sie die Google Container Registry des Projektes.
     - Funktioniert besser mit Cluster - Autoskalierung als mit manueller Knotenkonfiguration.
   - Auf einem Cluster bei dem die Knotenkonfiguration ungünstig ist können `imagePullSecrets` genutzt werden.
3. Cluster mit proprieritären Images, mit einigen Images die eine erweiterte Zugriffskontrolle erfordern.
   - Stellen sie sicher das AlwaysPullImages admission controller aktiv ist, anderenfalls können alle Pods potenziell Zugriff auf alle Images haben.
   - Verschieben sie sensitive Daten in eine "Secret" Ressource statt sie im Image abzulegen.
4. Ein mandantenfähiger Cluster in dem jeder Mandant eine eigene private Registry benötigt.
   - Stellen sie dicher das AlwaysPullImages admission controller aktiv ist, anderenfalls können alle Pods potenziell Zugriff auf alle Images haben.
   - Nutzen sie eine private Registry die eine Authorisierung erfordert.
   - Generieren die Registry - Zugriffsdaten für jeden Mandanten, abgelegt in einem Secret das in jedem Mandanten - Namespace vorhanden ist.
   - Der Mandant fügt dieses Sercret zu den imagePullSecrets in jedem seiner Namespace hinzu.

Falls die Zugriff auf mehrere Registries benötigen, können sie ein Secret für jede Registry erstellen, Kubelet wird jedwede `imagePullSecrets` in einer einzelnen `.docker/config.json` zusammenfassen.

4 - Workloads

4.1 - Pods

Pods sind die kleinsten einsetzbaren Einheiten, die in Kubernetes erstellt und verwaltet werden können.

Ein *Pod* (übersetzt Gruppe/Schote, wie z. B. eine Gruppe von Walen oder eine Erbsenschote) ist eine Gruppe von einem oder mehreren Containern mit gemeinsam genutzten Speicher- und Netzwerkressourcen und einer Spezifikation für die Ausführung der Container. Die Ressourcen eines Pods befinden sich immer auf dem gleichen (virtuellen) Server, werden gemeinsam geplant und in einem gemeinsamen Kontext ausgeführt. Ein Pod modelliert einen anwendungsspezifischen "logischen Server": Er enthält eine oder mehrere containerisierte Anwendungen, die relativ stark voneinander abhängen. In Nicht-Cloud-Kontexten sind Anwendungen, die auf demselben physischen oder virtuellen Server ausgeführt werden, vergleichbar zu Cloud-Anwendungen, die auf demselben logischen Server ausgeführt werden.

Ein Pod kann neben Anwendungs-Containern auch sogenannte Initialisierungs-Container enthalten, die beim Starten des Pods ausgeführt werden. Es können auch kurzlebige/ephemere Container zum Debuggen gestartet werden, wenn dies der Cluster anbietet.

Was ist ein Pod?

> **Hinweis:** Obwohl Kubernetes abgesehen von Docker auch andere Container-Laufzeitumgebungen unterstützt, ist Docker am bekanntesten und es ist hilfreich, Pods mit der Terminologie von Docker zu beschreiben.

Der gemeinsame Kontext eines Pods besteht aus einer Reihe von Linux-Namespaces, Cgroups und möglicherweise anderen Aspekten der Isolation, also die gleichen Dinge, die einen Dockercontainer isolieren. Innerhalb des Kontexts eines Pods können die einzelnen Anwendungen weitere Unterisolierungen haben.

Im Sinne von Docker-Konzepten ähnelt ein Pod einer Gruppe von Docker-Containern, die gemeinsame Namespaces und Dateisystem-Volumes nutzen.

Pods verwenden

Normalerweise müssen keine Pods erzeugt werden, auch keine Singleton-Pods. Stattdessen werden sie mit Workload-Ressourcen wie Deployment oder Job erzeugt. Für Pods, die von einem Systemzustand abhängen, ist die Nutzung von StatefulSet-Ressourcen zu erwägen.

Pods in einem Kubernetes-Cluster werden hauptsächlich auf zwei Arten verwendet:

- **Pods, die einen einzelnen Container ausführen**. Das "Ein-Container-per-Pod"-Modell ist der häufigste Kubernetes-Anwendungsfall. In diesem Fall kannst du dir einen einen Pod als einen Behälter vorstellen, der einen einzelnen Container enthält; Kubernetes verwaltet die Pods anstatt die Container direkt zu verwalten.
- **Pods, in denen mehrere Container ausgeführt werden, die zusammenarbeiten müssen**. Wenn eine Softwareanwendung aus co-lokaliserten Containern besteht, die sich gemeinsame Ressourcen teilen und stark voneinander abhängen, kann ein Pod die Container verkapseln. Diese Container bilden eine einzelne zusammenhängende Serviceeinheit, z. B. ein Container, der Daten in einem gemeinsam genutzten Volume öffentlich verfügbar macht, während ein separater *Sidecar*-Container die Daten aktualisiert. Der Pod fasst die Container, die Speicherressourcen und eine kurzlebiges Netzwerk-Identität als eine Einheit zusammen.

> **Hinweis:** Das Gruppieren mehrerer gemeinsam lokalisierter und gemeinsam verwalteter Container in einem einzigen Pod ist ein relativ fortgeschrittener Anwendungsfall. Du solltest diese Architektur nur in bestimmten Fällen verwenden, wenn deine Container stark voneinander abhängen.

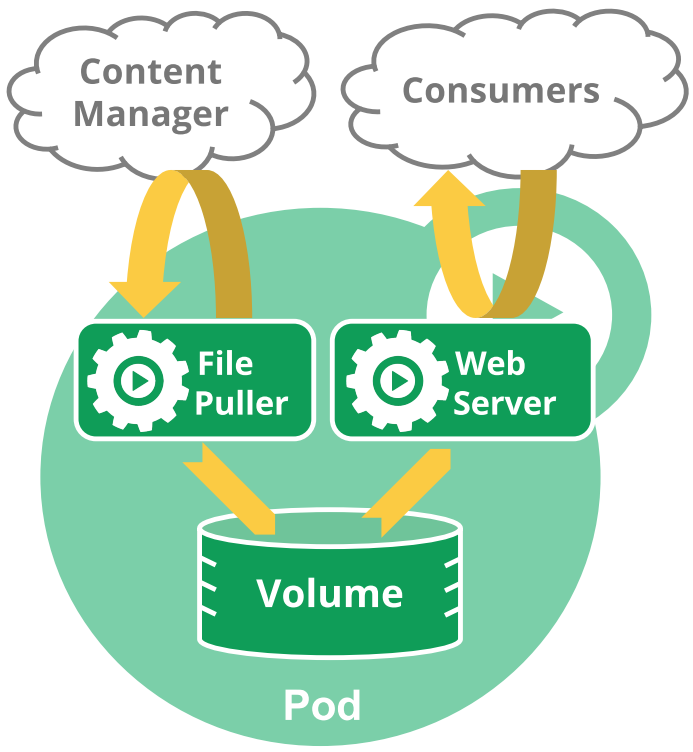
Jeder Pod sollte eine einzelne Instanz einer gegebenen Anwendung ausführen. Wenn du deine Anwendung horizontal skalieren willst (um mehr Instanzen auszuführen und dadurch mehr Gesamtressourcen bereitstellen), solltest du mehrere Pods verwenden, einen für jede Instanz. In Kubernetes wird dies typischerweise als Replikation bezeichnet. Replizierte Pods werden normalerweise als eine Gruppe durch eine Workload-Ressource und deren Controller erstellt und verwaltet.

Der Abschnitt Pods und Controller beschreibt, wie Kubernetes Workload-Ressourcen und deren Controller verwendet, um Anwendungen zu skalieren und zu heilen.

Wie Pods mehrere Container verwalten

Pods unterstützen mehrere kooperierende Prozesse (als Container), die eine zusammenhängende Serviceeinheit bilden. Kubernetes plant und stellt automatisch sicher, dass sich die Container in einem Pod auf demselben physischen oder virtuellen Server im Cluster befinden. Die Container können Ressourcen und Abhängigkeiten gemeinsam nutzen, miteinander kommunizieren und ferner koordinieren wann und wie sie beendet werden.

Zum Beispiel könntest du einen Container haben, der als Webserver für Dateien in einem gemeinsamen Volume arbeitet. Und ein separater "Sidecar" -Container aktualisiert die Daten von einer externen Datenquelle, siehe folgenden Abbildung:



Einige Pods haben sowohl Initialisierungs-Container als auch Anwendungs-Container. Initialisierungs-Container werden gestartet und beendet bevor die Anwendungs-Container gestartet werden.

Pods stellen standardmäßig zwei Arten von gemeinsam Ressourcen für die enthaltenen Container bereit: Netzwerk und Speicher.

Mit Pods arbeiten

Du wirst selten einzelne Pods direkt in Kubernetes erstellen, selbst Singleton-Pods. Das liegt daran, dass Pods als relativ kurzlebige Einweg-Einheiten konzipiert sind. Wann Ein Pod erstellt wird (entweder direkt von Ihnen oder indirekt von einem Controller), wird die Ausführung auf einem Node in Ihrem Cluster geplant. Der Pod bleibt auf diesem (virtuellen) Server, bis entweder der Pod die Ausführung beendet hat, das Pod-Objekt gelöscht wird, der Pod aufgrund mangelnder Ressourcen *evakuiert* wird oder oder der Node ausfällt.

> **Hinweis:** Das Neustarten eines Containers in einem Pod sollte nicht mit dem Neustarten eines Pods verwechselt werden. Ein Pod ist kein Prozess, sondern eine Umgebung zur Ausführung von Containern. Ein Pod bleibt bestehen bis er gelöscht wird.

Stelle beim Erstellen des Manifests für ein Pod-Objekt sicher, dass der angegebene Name ein gültiger DNS-Subdomain-Name ist.

Pods und Controller

Mit Workload-Ressourcen kannst du mehrere Pods erstellen und verwalten. Ein Controller für die Ressource kümmert sich um Replikation, Roll-Out sowie automatische Wiederherstellung im Fall von versagenden Pods. Wenn beispielsweise ein Node ausfällt, bemerkt ein Controller, dass die Pods auf dem Node nicht mehr laufen und plant die Ausführung eines Ersatzpods auf einem funktionierenden Node. Hier sind einige Beispiele für Workload-Ressourcen, die einen oder mehrere Pods verwalten:

- Deployment
- StatefulSet
- DaemonSet

Pod Vorlagen

Controller für Workload-Ressourcen erstellen Pods von einer *Pod Vorlage* und verwalten diese Pods für dich.

Pod Vorlagen sind Spezifikationen zum Erstellen von Pods und sind in Workload-Ressourcen enthalten wie z. B. Deployments, Jobs, and DaemonSets.

Jeder Controller für eine Workload-Ressource verwendet die Pod Vorlage innerhalb des Workload-Objektes, um Pods zu erzeugen. Die Pod Vorlage ist Teil des gewünschten Zustands der Workload-Ressource, mit der du deine Anwendung ausgeführt hast.

Das folgende Beispiel ist ein Manifest für einen einfachen Job mit einer `Vorlage`, die einen Container startet. Der Container in diesem Pod druckt eine Nachricht und pausiert dann.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
 name: hello
spec:
 template:
 # Dies is the Pod Vorlage
 spec:
 containers:
 - name: hello
 image: busybox
 command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
 restartPolicy: OnFailure
 # Die Pod Vorlage endet hier
```

Das Ändern der Pod Vorlage oder der Wechsel zu einer neuen Pod Vorlage hat keine direkten Auswirkungen auf bereits existierende Pods. Wenn du die Pod Vorlage für eine Workload-Ressource änderst, dann muss diese Ressource die Ersatz-Pods erstellen, welche die aktualisierte Vorlage verwenden.

Beispielsweise stellt der StatefulSet-Controller sicher, dass für jedes StatefulSet-Objekt die ausgeführten Pods mit der aktueller Pod Vorlage übereinstimmen. Wenn du das StatefulSet bearbeitest und die Vorlage änderst, beginnt das StatefulSet mit der Erstellung neuer Pods basierend auf der aktualisierten Vorlage. Schließlich werden alle alten Pods durch neue Pods ersetzt, und das Update ist abgeschlossen.

Jede Workload-Ressource implementiert eigenen Regeln für die Umsetzung von Änderungen der Pod Vorlage. Wenn du mehr über StatefulSet erfahren möchtest, dann lese die Seite Update-Strategien im Tutorial StatefulSet Basics.

Auf Nodes beobachtet oder verwaltet das Kubelet nicht direkt die Details zu Pod Vorlagen und Updates. Diese Details sind abstrahiert. Die Abstraktion und Trennung von Aufgaben vereinfacht die Systemsemantik und ermöglicht so das Verhalten des Clusters zu ändern ohne vorhandenen Code zu ändern.

Pod Update und Austausch

Wie im vorherigen Abschnitt erwähnt, erstellt der Controller neue Pods basierend auf der aktualisierten Vorlage, wenn die Pod Vorlage für eine Workload-Ressource geändert wird anstatt die vorhandenen Pods zu aktualisieren oder zu patchen.

Kubernetes hindert dich nicht daran, Pods direkt zu verwalten. Es ist möglich, einige Felder eines laufenden Pods zu aktualisieren. Allerdings haben Pod-Aktualisierungsvorgänge wie zum Beispiel `patch`, und `replace` einige Einschränkungen:

- Die meisten Metadaten zu einem Pod können nicht verändert werden. Zum Beispiel kannst du nicht die Felder `namespace`, `name`, `uid`, oder `creationTimestamp` ändern. Das `generation`-Feld muss eindeutig sein. Es werden nur Aktualisierungen akzeptiert, die den Wert des Feldes inkrementieren.
- Wenn das Feld `metadata.deletionTimestamp` gesetzt ist, kann kein neuer Eintrag zur Liste `metadata.finalizers` hinzugefügt werden.
- Pod-Updates dürfen keine Felder ändern, die Ausnahmen sind `spec.containers[*].image`, `spec.initContainers[*].image`, `spec.activeDeadlineSeconds` oder `spec.tolerations`. Für `spec.tolerations` kannnst du nur neue Einträge hinzufügen.
- Für `spec.activeDeadlineSeconds` sind nur zwei Änderungen erlaubt:
  1. ungesetztes Feld in eine positive Zahl
  2. positive Zahl in eine kleinere positive Zahl, die nicht negativ ist

Gemeinsame Nutzung von Ressourcen und Kommunikation

Pods ermöglichen den Datenaustausch und die Kommunikation zwischen den Containern, die im Pod enthalten sind.

Datenspeicherung in Pods

Ein Pod kann eine Reihe von gemeinsam genutzten Speicher- Volumes spezifizieren. Alle Container im Pod können auf die gemeinsamen Volumes zugreifen und dadurch Daten austauschen. Volumes ermöglichen auch, dass Daten ohne Verlust gespeichert werden, falls einer der Container neu gestartet werden muss. Im Kapitel Datenspeicherung findest du weitere Informationen, wie Kubernetes gemeinsam genutzten Speicher implementiert und Pods zur Verfügung stellt.

Pod-Netzwerk

Jedem Pod wird für jede Adressenfamilie eine eindeutige IP-Adresse zugewiesen. Jeder Container in einem Pod nutzt den gemeinsamen Netzwerk-Namespace, einschließlich der IP-Adresse und der Ports. In einem Pod (und **nur** dann) können die Container, die zum Pod gehören, über `localhost` miteinander kommunizieren. Wenn Container in einem Pod mit Entitäten *außerhalb des Pods* kommunizieren, müssen sie koordinieren, wie die gemeinsam genutzten Netzwerkressourcen (z. B. Ports) verwenden werden. Innerhalb eines Pods teilen

sich Container eine IP-Adresse und eine Reihe von Ports und können sich gegenseitig über `localhost` finden. Die Container in einem Pod können auch die üblichen Kommunikationsverfahren zwischen Prozessen nutzen, wie z. B. SystemV-Semaphoren oder "POSIX Shared Memory". Container in verschiedenen Pods haben unterschiedliche IP-Adressen und können nicht per IPC ohne spezielle Konfiguration kommunizieren. Container, die mit einem Container in einem anderen Pod interagieren möchten, müssen IP Netzwerke verwenden.

Für die Container innerhalb eines Pods stimmt der "hostname" mit dem konfigurierten `Namen` des Pods überein. Mehr dazu im Kapitel Netzwerke.

Privilegierter Modus für Container

Jeder Container in einem Pod kann den privilegierten Modus aktivieren, indem das Flag `privileged` im Sicherheitskontext der Container-Spezifikation verwendet wird. Dies ist nützlich für Container, die Verwaltungsfunktionen des Betriebssystems verwenden möchten, z. B. das Manipulieren des Netzwerk-Stacks oder den Zugriff auf Hardware. Prozesse innerhalb eines privilegierten Containers erhalten fast die gleichen Rechte wie sie Prozessen außerhalb eines Containers zur Verfügung stehen.

> **Hinweis:** Ihre Container-Umgebung muss das Konzept eines privilegierten Containers unterstützen, damit diese Einstellung relevant ist.

Statische Pods

Statische Pods werden direkt vom Kubelet-Daemon auf einem bestimmten Node verwaltet ohne dass sie vom API Server überwacht werden.

Die meisten Pods werden von der Kontrollebene verwaltet (z. B. Deployment). Aber für statische Pods überwacht das Kubelet jeden statischen Pod direkt (und startet ihn neu, wenn er ausfällt).

Statische Pods sind immer an ein Kubelet auf einem bestimmten Node gebunden. Der Hauptanwendungsfall für statische Pods besteht darin, eine selbst gehostete Steuerebene auszuführen. Mit anderen Worten: Das Kubelet dient zur Überwachung der einzelnen Komponenten der Kontrollebene.

Das Kubelet versucht automatisch auf dem Kubernetes API-Server für jeden statischen Pod einen spiegelbildlichen Pod (im Englischen: mirror pod) zu erstellen. Das bedeutet, dass die auf einem Node ausgeführten Pods auf dem API-Server sichtbar sind jedoch von dort nicht gesteuert werden können.

Nächste Schritte

- Verstehe den Lebenszyklus eines Pods.
- Erfahre mehr über RuntimeClass und wie du damit verschiedene Pods mit unterschiedlichen Container-Laufzeitumgebungen konfigurieren kannst.
- Mehr zum Thema Restriktionen für die Verteilung von Pods.
- Lese Pod-Disruption-Budget und wie du es verwenden kannst, um die Verfügbarkeit von Anwendungen bei Störungen zu verwalten. Die Pod -Objektdefinition beschreibt das Objekt im Detail.
- The Distributed System Toolkit: Patterns for Composite Containers erläutert allgemeine Layouts für Pods mit mehr als einem Container.

Um den Hintergrund zu verstehen, warum Kubernetes eine gemeinsame Pod-API in andere Ressourcen, wie z. B. StatefulSets oder Deployments einbindet, kannst du Artikel zu früheren Technologien lesen, unter anderem:

- Aurora

- Borg
- Marathon
- Omega
- Tupperware.

5 - Dienste, Lastverteilung und Netzwerkfunktionen

6 - Speicher

7 - Konfiguration

8 - Richtlinien

9 - Cluster Administration

9.1 - Proxies in Kubernetes

Auf dieser Seite werden die im Kubernetes verwendeten Proxies erläutert.

Proxies

Es gibt mehrere verschiedene Proxies, die die bei der Verwendung von Kubernetes begegnen können:

1. Der kubectl Proxy:
    - läuft auf dem Desktop eines Benutzers oder in einem Pod
    - Proxy von einer lokalen Host-Adresse zum Kubernetes API Server
    - Client zu Proxy verwendet HTTP
    - Proxy zu API Server verwendet HTTPS
    - lokalisiert den API Server
    - fügt Authentifizierungs-Header hinzu
2. Der API Server Proxy:
    - ist eine Bastion, die in den API Server eingebaut ist
    - verbindet einen Benutzer außerhalb des Clusters mit Cluster IPs, die sonst möglicherweise nicht erreichbar wären
    - läuft im API Server Prozess
    - Client zu Proxy verwendet HTTPS (oder http, wenn API Server so konfiguriert ist)
    - Proxy zum Ziel kann HTTP oder HTTPS verwenden, der Proxy wählt dies unter Verwendung der verfügbaren Informationen aus
    - kann verwendet werden, um einen Knoten, Pod oder Service zu erreichen
    - führt einen Lastausgleich durch um einen Service zu erreichen, wenn dieser verwendet wird
3. Der kube Proxy:
    - läuft auf jedem Knoten
    - Proxy unterstüzt UDP, TCP und SCTP
    - versteht kein HTTP
    - stellt Lastausgleich zur Verfügung
    - wird nur zum erreichen von Services verwendet
4. Ein Proxy/Load-balancer vor dem API Server:
    - Existenz und Implementierung variieren von Cluster zu Cluster (z.B. nginx)
    - sitzt zwischen allen Clients und einem oder mehreren API Servern
    - fungiert als Load Balancer, wenn es mehrere API Server gibt
5. Cloud Load Balancer für externe Services:
    - wird von einigen Cloud Anbietern angeboten (z.B. AWS ELB, Google Cloud Load Balancer)
    - werden automatisch erstellt, wenn der Kubernetes Service den Typ `LoadBalancer` hat
    - unterstützt normalerweiße nur UDP/TCP
    - Die SCTP-Unterstützung hängt von der Load Balancer Implementierung des Cloud Providers ab
    - die Implementierung variiert je nach Cloud Anbieter

Kubernetes Benutzer müssen sich in der Regel um nichts anderes als die ersten beiden Typen kümmern. Der Cluster Administrator stellt in der Regel sicher, dass die letztgenannten Typen korrekt eingerichtet sind.

Anforderung an Umleitungen

Proxies haben die Möglichkeit der Umleitung (redirect) ersetzt. Umleitungen sind veraltet.

9.2 - Controller Manager Metriken

Controller Manager Metriken liefern wichtige Erkenntnisse über die Leistung und den Zustand von den Controller Managern.

Was sind Controller Manager Metriken

Die Kennzahlen des Controller Managers liefert wichtige Erkenntnisse über die Leistung und den Zustand des Controller Managers. Diese Metriken beinhalten gängige Go Language Laufzeitmetriken wie go_routine count und controller-spezifische Metriken wie z.B. etcd Request Latenzen oder Cloud Provider (AWS, GCE, OpenStack) API Latenzen, die verwendet werden können um den Zustand eines Clusters zu messen.

Ab Kubernetes 1.7 stehen detaillierte Cloud Provider Metriken für den Speicherbetrieb für GCE, AWS, Vsphere und OpenStack zur Verfügung. Diese Metriken können verwendet werden, um den Zustand persistenter Datenträgeroperationen zu überwachen.

Für GCE werden diese Metriken beispielsweise wie folgt aufgerufen:

```
cloudprovider_gce_api_request_duration_seconds { request = "instance_list"}
cloudprovider_gce_api_request_duration_seconds { request = "disk_insert"}
cloudprovider_gce_api_request_duration_seconds { request = "disk_delete"}
cloudprovider_gce_api_request_duration_seconds { request = "attach_disk"}
cloudprovider_gce_api_request_duration_seconds { request = "detach_disk"}
cloudprovider_gce_api_request_duration_seconds { request = "list_disk"}
```

Konfiguration

In einem Cluster sind die Controller Manager Metriken unter `http://localhost:10252/metrics` auf dem Host verfügbar, auf dem der Controller Manager läuft.

Die Metriken werden im Prometheus Format ausgegeben.

In einer Produktionsumgebung können Sie Prometheus oder einen anderen Metrik Scraper konfigurieren, um diese Metriken regelmäßig zu sammeln und in einer Art Zeitreihen Datenbank verfügbar zu machen.

9.3 - Addons Installieren

Add-Ons erweitern die Funktionalität von Kubernetes.

Diese Seite gibt eine Übersicht über einige verfügbare Add-Ons und verweist auf die entsprechenden Installationsanleitungen.

Die Add-Ons in den einzelnen Kategorien sind alphabetisch sortiert - Die Reihenfolge impliziert keine bevorzugung einzelner Projekte.

Networking und Network Policy

- ACI bietet Container-Networking und Network-Security mit Cisco ACI.
- Calico ist ein Networking- und Network-Policy-Provider. Calico unterstützt eine Reihe von Networking-Optionen, damit Du die richtige für deinen Use-Case wählen kannst. Dies beinhaltet Non-Overlaying and Overlaying-Networks mit oder ohne BGP. Calico nutzt die gleiche Engine um Network-Policies für Hosts, Pods und (falls Du Istio & Envoy benutzt) Anwendungen auf Service-Mesh-Ebene durchzusetzen.
- Canal vereint Flannel und Calico um Networking- und Network-Policies bereitzustellen.
- Cilium ist ein L3 Network- and Network-Policy-Plugin welches das transparent HTTP/API/L7-Policies durchsetzen kann. Sowohl Routing- als auch Overlay/Encapsulation-Modes werden uterstützt. Außerdem kann Cilium auf andere CNI-Plugins aufsetzen.
- CNI-Genie ermöglicht das nahtlose Verbinden von Kubernetes mit einer Reihe an CNI-Plugins wie z.B. Calico, Canal, Flannel, Romana, oder Weave.
- Contiv bietet konfigurierbares Networking (Native L3 auf BGP, Overlay mit vxlan, Klassisches L2, Cisco-SDN/ACI) für verschiedene Anwendungszwecke und auch umfangreiches Policy-Framework. Das Contiv-Projekt ist vollständig Open Source. Der installer bietet sowohl kubeadm als auch nicht-kubeadm basierte Installationen.
- Contrail, basierend auf Tungsten Fabric, ist eine Open Source, multi-Cloud Netzwerkvirtualisierungs- und Policy-Management Plattform. Contrail und Tungsten Fabric sind mit Orechstratoren wie z.B. Kubernetes, OpenShift, OpenStack und Mesos integriert und bieten Isolationsmodi für Virtuelle Maschinen, Container (bzw. Pods) und Bare Metal workloads.
- Flannel ist ein Overlay-Network-Provider der mit Kubernetes genutzt werden kann.
- Knitter ist eine Network-Lösung die Mehrfach-Network in Kubernetes ermöglicht.
- Multus ist ein Multi-Plugin für Mehrfachnetzwerk-Unterstützung um alle CNI-Plugins (z.B. Calico, Cilium, Contiv, Flannel), zusätzlich zu SRIOV-, DPDK-, OVS-DPDK- und VPP-Basierten Workloads in Kubernetes zu unterstützen.
- NSX-T Container Plug-in (NCP) bietet eine Integration zwischen VMware NSX-T und einem Orchestator wie z.B. Kubernetes. Außerdem bietet es eine Integration zwischen NSX-T und Containerbasierten CaaS/PaaS-Plattformen wie z.B. Pivotal Container Service (PKS) und OpenShift.
- Nuage ist eine SDN-Plattform die Policy-Basiertes Networking zwischen Kubernetes Pods und nicht-Kubernetes Umgebungen inklusive Sichtbarkeit und Security-Monitoring bereitstellt.
- Romana ist eine Layer 3 Network-Lösung für Pod-Netzwerke welche auch die NetworkPolicy API unterstützt. Details zur Installation als kubeadm Add-On sind hier verfügbar.
- Weave Net bietet Networking and Network-Policies und arbeitet auf beiden Seiten der Network-Partition ohne auf eine externe Datenbank angwiesen zu sein.

Service-Discovery

- CoreDNS ist ein flexibler, erweiterbarer DNS-Server der in einem Cluster installiert werden kann und das Cluster-interne DNS für Pods bereitzustellen.

Visualisierung & Überwachung

- Dashboard ist ein Dashboard Web Interface für Kubernetes.
- Weave Scope ist ein Tool um Container, Pods, Services usw. Grafisch zu visualieren. Kann in Verbindung mit einem Weave Cloud Account genutzt oder selbst gehosted werden.

Infrastruktur

- KubeVirt ist ein Add-On um Virtuelle Maschinen in Kubernetes auszuführen. Wird typischer auf Bare-Metal Clustern eingesetzt.

Legacy Add-Ons

Es gibt einige weitere Add-Ons die in dem abgekündigten cluster/addons-Verzeichnis dokumentiert sind.

Add-Ons die ordentlich gewartet werden dürfen gerne hier aufgezählt werden. Wir freuen uns auf PRs!

10 - Kubernetes erweitern

11 - Konzept Dokumentations-Vorlage

> **Hinweis:** Stellen Sie auch sicher einen Eintrag im Inhaltsverzeichnis für Ihr neues Dokument zu erstellen.

Diese Seite erklärt ...

Verstehen ...

Kubernetes bietet ...

Verwenden ...

Benutzen Sie ...

Nächste Schritte

[Optionaler Bereich]

- Lernen Sie mehr über ein neues Thema schreiben.
- Besuchen Sie Seitenvorlagen verwenden - Konzeptvorlage wie Sie diese Vorlage verwenden.