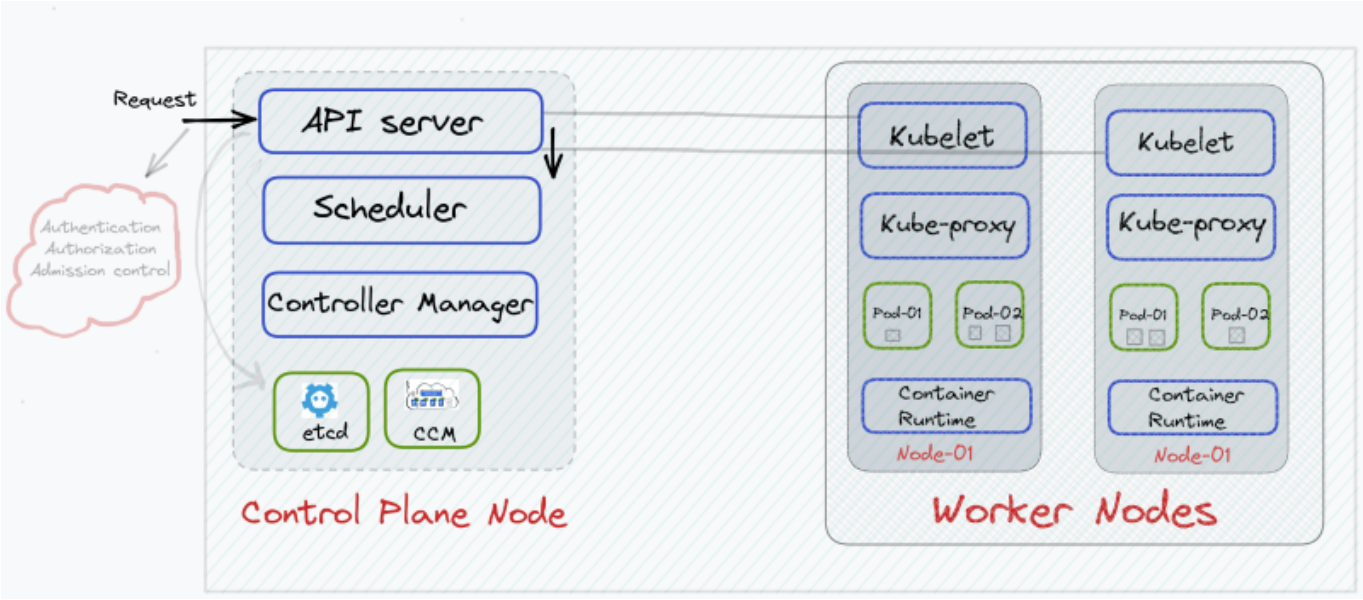
Verständnis der Architektur von Kubernetes

Kubernetes ist ein leistungsstarkes System zur Orchestrierung von Containern, das die Welt des Cloud Computings im Sturm erobert hat. Mit seiner Fähigkeit, Container über mehrere Hosts hinweg zu verwalten und zu skalieren, ist Kubernetes zur bevorzugten Plattform geworden, wenn es darum geht, containerisierte Anwendungen im Produktionsumfeld auszuführen. In diesem Blogbeitrag schauen wir uns die Architektur von Kubernetes genauer an und wie es funktioniert, um containerisierte Anwendungen zu verwalten und zu skalieren.

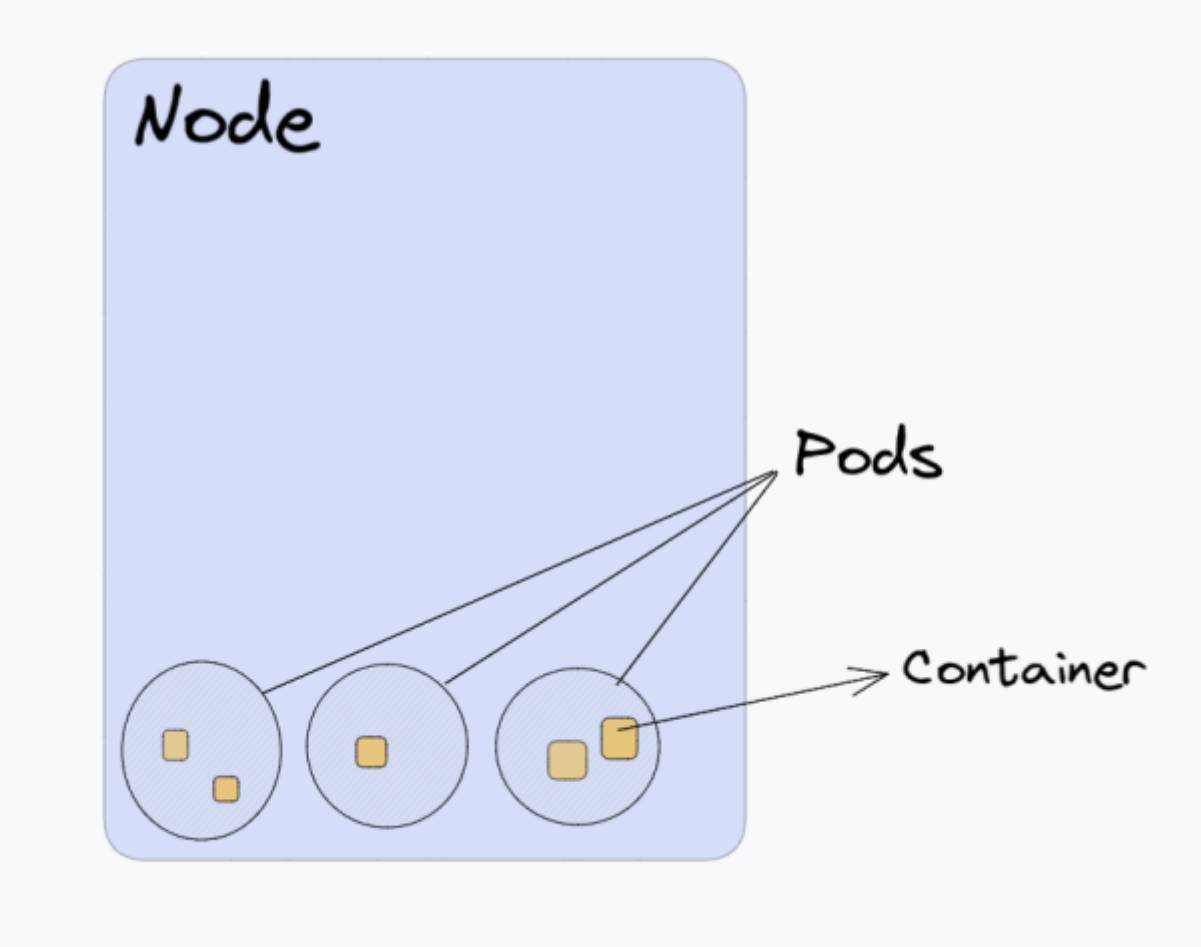
Sieh Dir die folgende Grafik einmal genauer an (insbesondere die Pfeile und Linien). Wir werden gemeinsam untersuchen, was genau diese Pfeile darstellen, was ein API-Server ist, was es mit der Linie auf sich hat, die vom Kubelet zum API-Server zeigt, was der Unterschied zwischen der Control Plane und einem Worker Node ist, warum eine Control Plane überhaupt benötigt wird und viele weitere Fragen wie diese werden bis zum Ende dieses Blogs vollständig beantwortet sein.



Bevor du die Architektur verstehst, lass uns zunächst die Begriffe betrachten, die im Kubernetes-Umfeld weit verbreitet sind.

1. **Node:** Ein Node (Knoten) in Kubernetes ist die Darstellung einer einzelnen Maschine in deinem Kubernetes-Cluster. In einer Produktionsumgebung ist ein Node höchstwahrscheinlich eine physische Maschine in einem Rechenzentrum oder eine virtuelle Maschine in der Cloud.
2. **Pods:** Kubernetes führt Container nicht direkt aus. Stattdessen werden die Container in einer kugelförmigen Einheit namens „Pod“ ausgeführt.

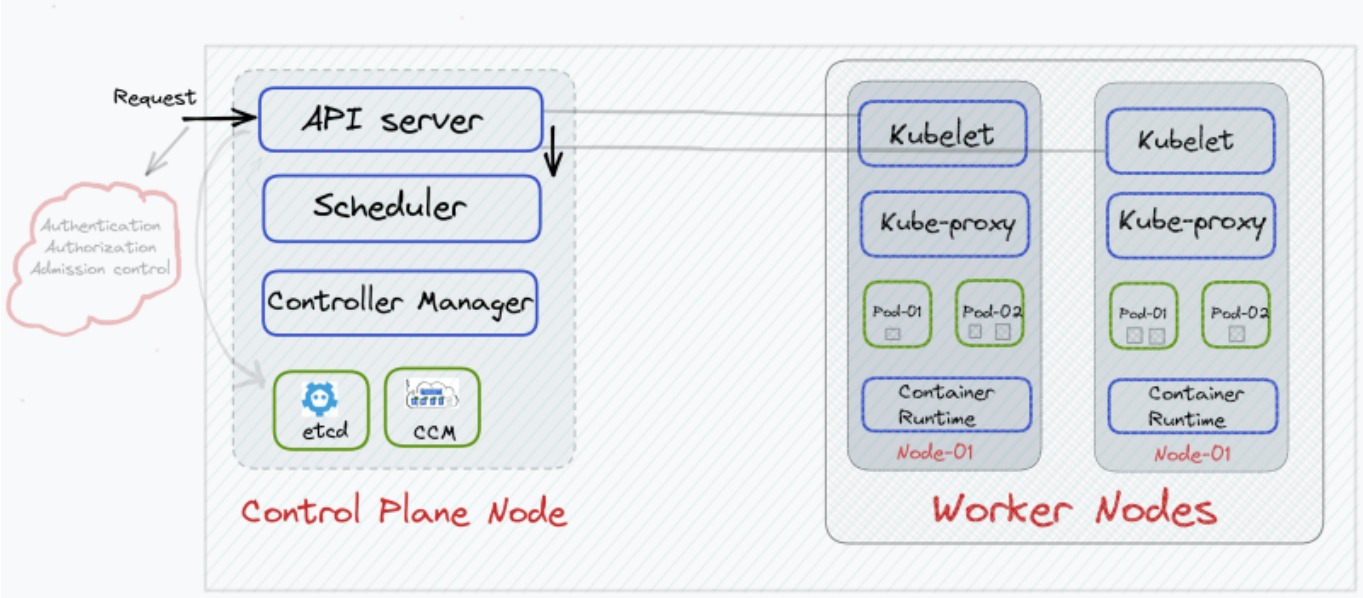
- Innerhalb deines Nodes läuft der Pod. Innerhalb deines Pods laufen die Container. Und innerhalb deiner Container läuft deine Anwendung.



Cluster: Ein Cluster ist eine Gruppe aus Nodes, sowohl physisch als auch virtuell, die dazu dient, containerisierte Anwendungen auszuführen.

- Angenommen, du hast drei Nodes, die verwendet werden, um eine containerisierte Anwendung auszuführen. Die Gruppierung all dieser Nodes zu einer leistungsfähigeren Einheit ist das, was man als Cluster bezeichnet.

Lass uns versuchen, die Architektur von Kubernetes besser zu verstehen – also was „hinter den Kulissen“ passiert.



Sieh dir das obige Bild genau an. Wir haben drei Nodes gruppiert (1 Control Plane & 2 Worker Nodes), was man zusammen als Kubernetes-Cluster bezeichnet. Du kannst so viele Nodes erstellen, wie du möchtest – je nach Anforderung der Anwendung.

In jedem Kubernetes-Cluster gibt es zwei Arten von Nodes:

1. Einen oder mehrere **Control Plane Nodes** (auch Master Nodes genannt)
2. Einen oder mehrere **Worker Nodes**

Wenn du dir das oben ansiehst, erkennst du, dass jeder Node bestimmte Komponenten enthält. Der Control Plane Node hat Komponenten wie den API-Server, Scheduler usw., während die Worker Nodes Komponenten wie kubelet, kube-proxy usw. enthalten. Wir werden uns gleich die jeweilige Rolle jeder Komponente in einem Node anschauen.

Die Worker Nodes sind dafür zuständig, deine Anwendung auszuführen (du siehst ja, dass sich die Pods im Worker Node befinden), während der Control Plane Node für die Verwaltung deines Clusters zuständig ist: Cluster starten, Nodes hinzufügen, Pods entfernen, Pods skalieren und vieles mehr.

Ohne einen Control Plane Node kann dein Kubernetes-Cluster nicht funktionieren. Es ist also entscheidend, dass der Control Plane durchgehend läuft.

Lass uns die Funktionsweise jeder Control-Plane-Komponente genauer anschauen:

1. **API-Server:** Der API-Server ist das „Gehirn“ hinter allen Vorgängen im Kubernetes-Cluster. Um mit dem Cluster zu interagieren, werden alle Anfragen an den API-Server gesendet. Der API-Server empfängt REST-Anfragen von Benutzern, Administratoren, Entwicklern, Operatoren und externen Tools, prüft und verarbeitet sie. Immer wenn du eine Anfrage an den API-Server stellst, durchläuft sie drei Schritte:
    - Authentifizierung: Stellt sicher, dass du bist, wer du vorgibst zu sein.
    - Autorisierung: Prüft, ob du berechtigt bist, die Aktion durchzuführen (z. B. über RBAC).
    - Admission-Control: Wendet Regelwerke auf Pods an, bevor sie ausgeführt werden.
2. **Scheduler:** Die Hauptaufgabe des Schedulers ist es, Pods den passenden Nodes zuzuweisen. Zum Beispiel: Du möchtest per Befehlszeile einen Pod starten – dann wird diese Anfrage verarbeitet, authentifiziert, autorisiert und durch Admission Control überprüft. Danach geht sie an den Scheduler, der dann den besten verfügbaren Worker Node für diesen Pod auswählt. Der Scheduler bestimmt gültige Nodes zur Platzierung eines Pods, bewertet sie anhand verfügbarer

und benötigter Ressourcen und weist schließlich dem Pod einen bestimmten Node zu.
3. **Controller Manager:** Diese Komponente kümmert sich um die Einhaltung des gewünschten Zustands deines Kubernetes-Clusters durch sogenannte Controller oder Operatoren. Controller sind Prozesse, die in einer Dauerschleife laufen, und den tatsächlichen Zustand des Clusters mit dem gewünschten Zustand vergleichen. Aber wo wird der tatsächliche Zustand gespeichert? Im etcd-Store.

Key-Value-Datenbank (etcd): Alle Informationen zum Cluster werden in etcd gespeichert. Wichtig: Anwendungsdaten („App-Daten“) werden nicht in etcd abgelegt. Neue Daten im etcd-Store werden immer angehängt, niemals überschrieben. Veraltete/unerwünschte Daten werden regelmäßig entfernt, um den Speicher klein zu halten. etcd verwendet den Raft-Konsensalgorithmus. Erinnerst du dich noch, wie oben erwähnt wurde, dass der Scheduler einen passenden Node für den Pod auswählt? Hier ist, wie das tatsächlich abläuft:

- Eine Anfrage wird vom Client an den API-Server geschickt, um einen Pod zu starten.
- Der API-Server validiert die Benutzeridentität & Anfrage.
- Die Anfrage wird an den Scheduler weitergeleitet.
- Der Scheduler fragt beim API-Server aktuelle Cluster-Information
- Der API-Server ist die einzige Komponente, die Daten im etcd-Speicher lesen und schreiben kann. Keine andere Komponente kann direkt mit dem etcd-Store kommunizieren.
- Nachdem der API-Server die Informationen erhalten hat, informiert er den Scheduler.
- Basierend auf den Informationen, die der API-Server erhalten hat, weist der Scheduler den Pod einem Node zu.

Jetzt fragst Du mich vielleicht: „Heißt das, dass der Pod jetzt läuft?“

Die Antwort ist: Nein.

Bis zu diesem Zeitpunkt wurde lediglich der richtige Node ausgewählt, auf dem ein Pod laufen soll – aber der Pod läuft noch nicht.

1. **Cloud Controller Manager (CCM):** Der CCM ist dafür zuständig, Controller oder Operatoren auszuführen, um mit der zugrunde liegenden Infrastruktur eines Cloud-Host-Anbieters zu kommunizieren, wenn Nodes nicht verfügbar sind.

Schau Dir jetzt nochmal alle Komponenten der Control Plane in Ruhe an, bevor Du weitermachst.

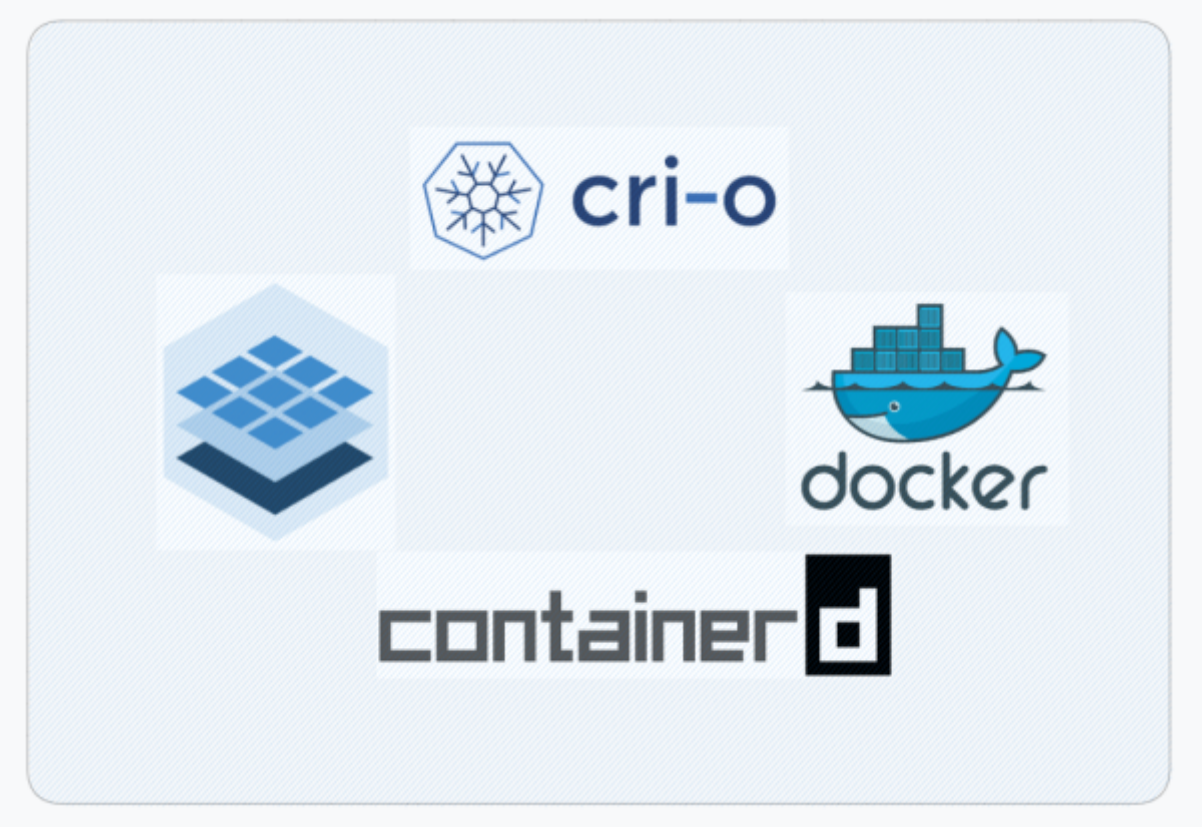
Jetzt ist es an der Zeit, die Bestandteile eines Worker-Nodes besser zu verstehen:

Ein Worker-Node stellt die Umgebung bereit, in der eine containerisierte Anwendung ausgeführt wird. Die Komponenten, die sich auf einem Worker-Node befinden, sind:

- Kubelet
- kube-proxy
- Container-Runtime
- Addons oder DNS

1. **Container-Runtime:** Auch wenn Kubernetes als „Tool zur Container-Orchestrierung“ gilt, kann es Container nicht direkt selbst ausführen. Deshalb wird auf jedem Node, auf dem ein Pod geplant ist, eine Container-Runtime benötigt, um den Lebenszyklus der Container zu verwalten. **Wichtig: Eine Container-Runtime ist sowohl auf den Nodes der Control Plane als auch auf den Worker-Nodes vorhanden.** Kubernetes unterstützt mehrere Container-Runtimes,

darunter:
- CRI-O
- containerd
- Docker
- Mirantis Container Runtime



2. **Kubelet:** Genau wie die Container-Laufzeitumgebung ist das Kubelet sowohl auf den Steuerungsknoten (Control Plane) als auch auf dem Arbeitsknoten (Worker Node) vorhanden. Um ein Pod auf einem Arbeitsknoten auszuführen, braucht es eine Komponente, die mit der Steuerungsebene kommuniziert, damit das Pod gestartet werden kann. Das Kubelet ist genau diese Komponente. Das Kubelet jedes Knotens kommuniziert mit der Steuerungsebene und wartet auf den Befehl vom API-Server, ein Pod zu starten. Sobald das Kubelet eines Knotens den Befehl vom API-Server erhält, kommuniziert es über eine Plugin-basierte Schnittstelle (CRI-Shim) mit der Container-Laufzeitumgebung seines Knotens. Und damit beginnt das Pod zu laufen.

Falls Du jetzt ein bisschen verwirrt bist, hier ist die Zusammenfassung der internen Funktionsweise der Kubernetes-Komponenten: **Das solltest Du nicht verpassen!**

- Der Nutzer (also Du) sendet eine Anfrage an den API-Server, um ein Pod zu starten.

```
kubectl run <pod-name> --image=<image-name>
```

- Diese Anfrage wird jetzt vom API-Server validiert.
- Der API-Server leitet diese Anfrage an den Scheduler auf der Steuerungsebene (Control Plane) weiter.
- Im Gegenzug fordert der Scheduler clusterbezogene Informationen vom API-Server an, da der API-Server die einzige Komponente ist, die mit dem etcd-Store interagieren kann.
- Nachdem der API-Server diese Anfrage vom Scheduler erhalten hat, liest er die Daten aus dem etcd-Store und stellt sie dem Scheduler zur Verfügung.
- Der Scheduler weist daraufhin auf Grundlage der erhaltenen Informationen einem Node ein Pod

zu und teilt diese Entscheidung dem API-Server mit.
„Hey API-Server, das Pod soll auf node-01 laufen.“
– Scheduler

- Der API-Server weist dem Kubelet des entsprechenden Nodes die Aufgabe zu, das Pod zu starten.
- Nachdem der Kubelet des Nodes die Anweisung vom API-Server erhalten hat, interagiert er über den CRI-Shim mit der Container Runtime – und jetzt läuft das Pod auf einem bestimmten Node.
- Während das Pod läuft, überprüft der Controllermanager, ob der gewünschte Zustand des Clusters dem tatsächlichen Zustand des Kubernetes-Clusters entspricht.