Der Begriff „**Skalierung**“ beschreibt, wie einfach sich eine Software bei einem veränderten Ressourcenverbrauch anpassen lässt. In der Praxis wird zwischen zwei Arten der Skalierung unterschieden:

1. **Vertikale Skalierung**: Der gestiegene Ressourcenverbrauch wird durch Anpassung der Hardware kompensiert. Das bedeutet zum Beispiel, dass ein alter Server durch einen leistungsstärkeren neuen Server ersetzt wird. Insgesamt steigt die Verarbeitungslast des Servers durch die Erhöhung der Anfragenanzahl (Scale-up).

2. **Horizontale Skalierung**: Der gestiegene Ressourcenverbrauch wird auf mehrere Instanzen verteilt. Das geschieht beispielsweise, wenn anstelle eines Servers nun 2 Server Anfragen verarbeiten können. Dadurch sinkt die Verarbeitungslast jedes einzelnen Servers (Scale-down).

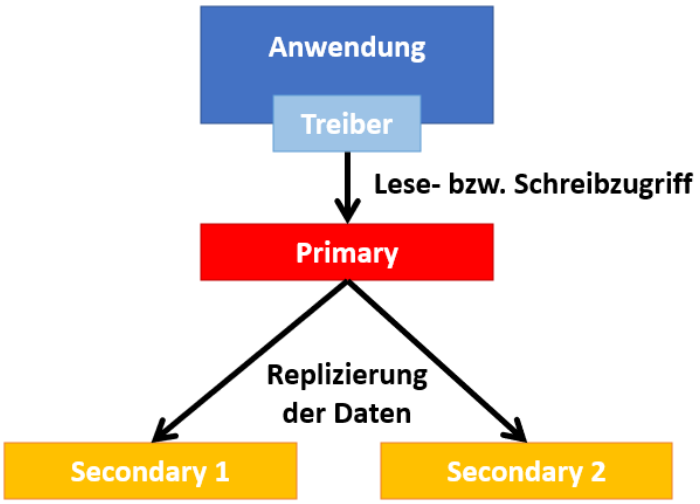
Aus wirtschaftlicher Sicht ist eine horizontale Skalierung vorzuziehen, da diese in aller Regel kostengünstiger ist. Zudem ist die horizontale Skalierung flexibler. Sollte sich der Ressourcenverbrauch verändern, kann auf die Veränderung entsprechend reagiert werden. Die Möglichkeiten der vertikale Skalierung sind darüber hinaus auf die Grenzen des derzeitig technisch Machbarem beschränkt.

Eine horizontale Skalierung wird heute verstärkt im Rahmen des Cloud-Computings realisiert. **Database as a Service** (DBaaS) bezeichnet die Möglichkeit, die Anlegung von Datenbanken mit spezifischen Eigenschaften bei Providern in Auftrag zu geben. Der Provider stellt die Hardware (z. B. Serverfarmen oder Rechenzentren) und Software (u.a. die gewünschte Datenbank) bereit. Darüber hinaus kümmert er sich um die Wartung der Hardware und Software. Der Benutzer erhält die Möglichkeit, die Funktionsfähigkeit der Datenbank zu überwachen und die Rechenkapazitäten bei Bedarf per Auftrag zu vergrößern. Techniken, die im Bereich des Cloud-Computings zur Gewährleistung der Ausfallsicherheit und der Skalierbarkeit angewandt werden, sind Cluster, Replica Sets und Sharding. Sie werden im Folgenden kurz anhand der Umsetzung in der NoSQL-Datenbank MongoDB Atlas erläutert:

Ein **Cluster** (deutsch: Schwarm) bezeichnet einen Verbund aus mindestens 2 Rechnern, Storages oder anderen Hardwarekomponenten, die parallel laufen und ein gemeinsames Ziel verfolgen. Das Ziel kann beispielsweise die Bereitstellung einer MongoDB als Service durch einen Provider sein. Aufgaben, die parallel abgearbeitet werden können, werden im Rahmen der Arbeitsteilung im Verbund verteilt. Dieses Vorgehen reduziert die Arbeitslast der einzelnen Komponenten. Mit einem Cluster können somit die Verfügbarkeit, die Rechenkapazität und die Speicherkapazität erhöht werden.

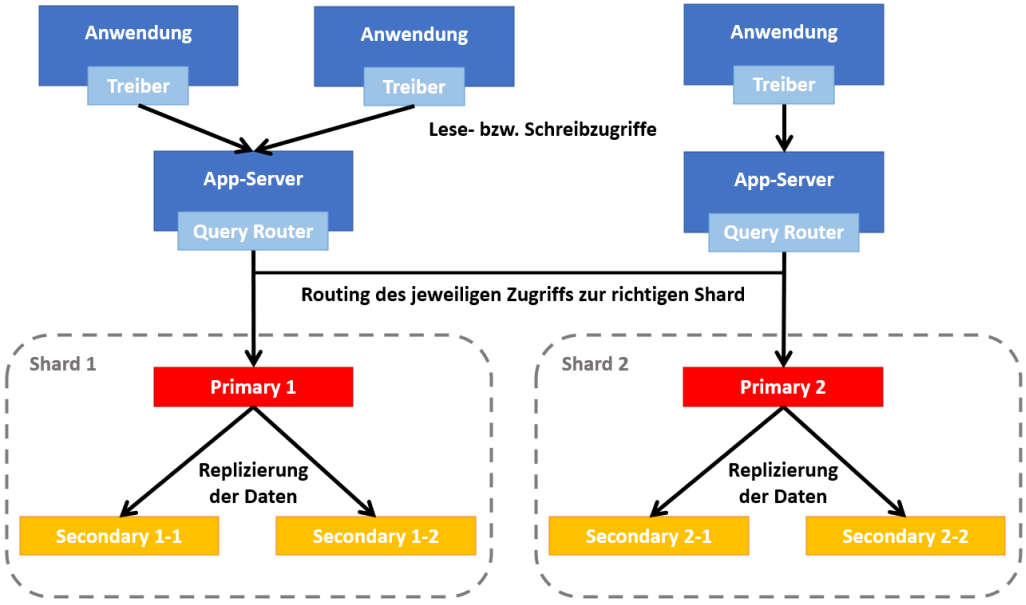
Ein **Replica Set** besteht aus einer Gruppe von einem oder mehreren Servern oder mehreren Prozessen, die Kopien von gespeicherten Daten einer Datenbank verwalten. Das Replica Set wird in der MongoDB in Knoten unterteilt und kann auf einem Cluster laufen. Es werden 2 Arten von Knoten unterschieden:

1. Der **Primary** ist ein Knoten, der auch als Masterknoten bezeichnet wird. Jedes Replica Set kann nur einen Primary besitzen. Bei dem Primary kommen alle Lese- und Schreibzugriffe an. Diese wendet er auf die bei ihm gespeicherten Datensätze an. Darüber hinaus repliziert (= sichern / kopieren) er die bei ihm gespeicherten Daten auf Secondaries.



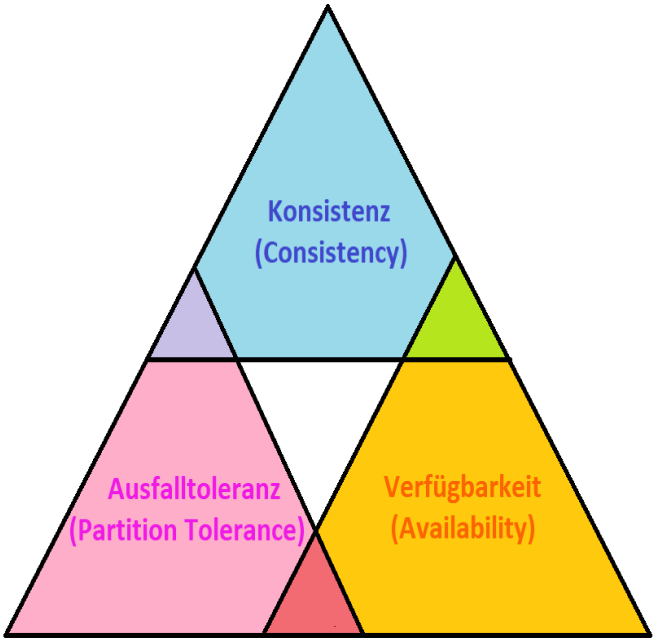
2. Der **Secondary** ist ein Knoten, der standardmäßig keine Lese- und Schreibrechte besitzt. Er verwaltet im Regelfall ausschließlich Kopien der Daten des Primaries. Es wird empfohlen, dass jedes Replica Set mindestens 2 Secondaries besitzt. Sollte der Primary ausfallen, wählen alle Secondaries eines Replica Sets einen Secondary zum neuen Primary. Somit tragen die Secondaries zur Erhöhung der Ausfallsicherheit bei.

Das **Sharding** ist eine Möglichkeit, eine MongoDB-Datenbank in einem Cluster horizontal zu skalieren. Die Technik wird eingesetzt, wenn die Datenmenge, die in der Datenbank gespeichert werden muss, zu groß wird. Beim Sharding werden Datensätze auf mehrere Hosts bzw. Instanzen (= **Shards**, engl. Scherben) verteilt. Auf diese Weise wird die Datenbank zu einem verteilten System. Shards können ihrerseits durch Replica Sets vor Ausfällen abgesichert werden. Damit Lese- und Schreibzugriffe auf die Datenbank zur richtigen Shard gelangen, gibt es einen oder mehrere Query Router. Ein Query Router stellt den Zugangspunkt zum Datenbank-Cluster dar und agiert bei Anfragen von Anwendungen an die Datenbank wie ein Proxy. Er entscheidet, welche Shards die Anfrage ausführen müssen, und leitet sie entsprechend weiter.



Durch Cloud-Computing mit horizontaler Skalierung werden Datenbanken zu verteilten Systemen. Dies führt unweigerlich zum **CAP-Theorem**, das besagt, dass nur 2 der 3 Eigenschaften Konsistenz (engl. Consistency), Verfügbarkeit (engl. Availability) und Ausfalltoleranz (engl. Partition Tolerance) in einem verteilten System gleichzeitig realisiert werden können. Die 3 Eigenschaften werden für das CAP-Theorem wie folgt definiert:

Konsistenz: Nach Abschluss einer Transaktion befindet sich das System in einem konsistenten Zustand. Das bedeutet beispielsweise, dass alle Replizierungen in Replication Sets abgeschlossen sein müssen, bevor eine neue Transaktion ausgeführt werden kann. Damit wird sichergestellt, dass das verteilte System immer mit den neuesten Daten arbeitet.



Verfügbarkeit: Die Reaktionszeit auf eine Anfrage muss angemessen sein. Damit wird sichergestellt, dass jede Anfrage beantwortet wird. Welche Reaktionszeit als „angemessen“ gilt, unterscheidet sich je nach Einsatzzweck des verteilten Systems.

Ausfalltoleranz (auch: **Partitionstoleranz**): Das verteilte System arbeitet weiter, auch wenn ein Teil des Systems ausfällt. Dies kann zum Beispiel über Replica Sets realisiert werden, in denen ausgefallene Primaries durch Secondaries ersetzt werden.

Bei der Realisierung einer Datenbank als verteiltes System sollten die fachlichen Anforderungen genutzt werden, um eine Priorisierung im Hinblick auf die 3 Eigenschaften vorzunehmen. Auf Grundlage der Priorisierung können anschließend konkrete Umsetzungsmaßnahmen beschlossen werden.

Fragen zum Text

1. Welche Vorteile hat die Verwendung einer Cloud zum Hosten der Datenbank für Herrn Nettmann im Gegensatz zum eigenen Datenbankserver?

2. Eignet sich in einem verteilten System eine NoSQL-Datenbank wie MongoDB für eine Speicherung der Teilsystem Zustände (Pumpe an/aus oder Werkstatttore geschlossen) im Autohaus Nettmann?

3. Welche Vorteile im Hinblick auf die Skalierung sind bei einer NoSQL Datenbanken (z. B. MongoDB) im Gegensatz zu einer relationalen Datenbank gegeben?

