Linux-Container-Architektur

Was hat man sich bloß dabei gedacht?

Jörg Kastning

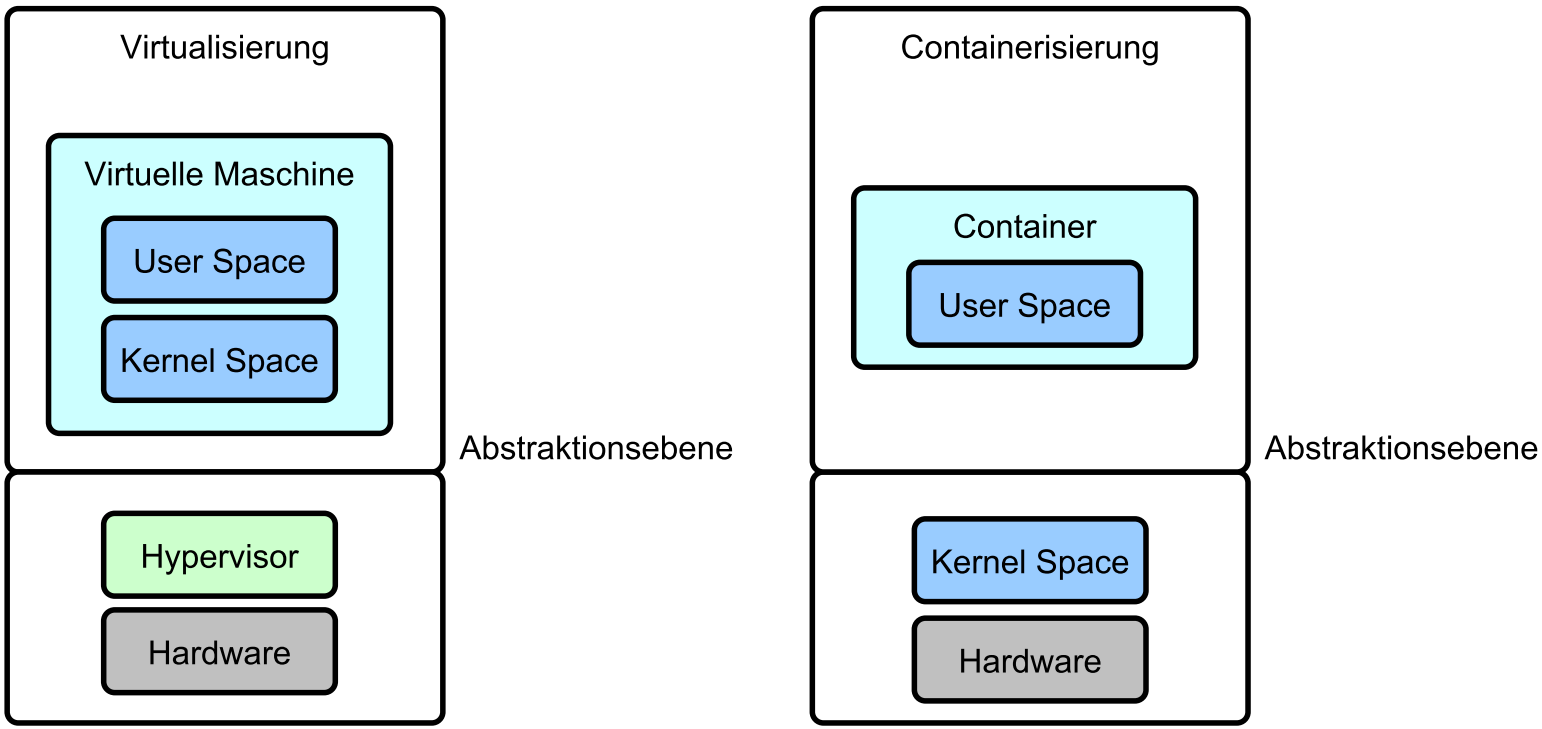
Universität Bielefeld / BITS

22. Juni 2021

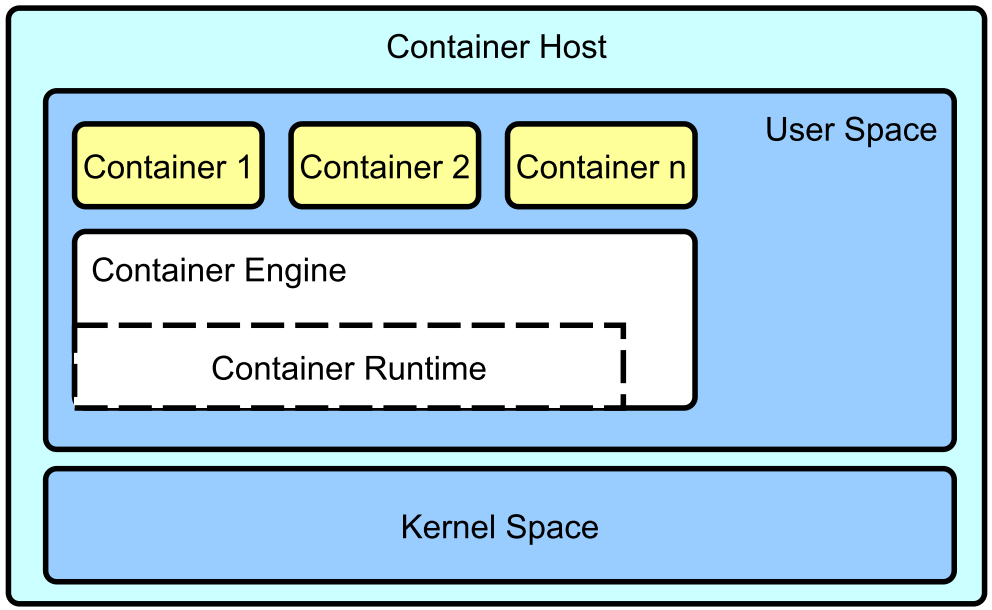
Warum sind wir hier?

- Linux-Container sind seit ca. 2015 das Buzzword neben KI und Machine Learning.
- Sie versprechen (wie immer) die Lösung all unserer Probleme zu sein.
- Zeit sie zu entzaubern und zu prüfen, wo sie wirklich von Nutzen sein können.

Virtuelle Maschinen vs. Container



Container-Virtualisierung

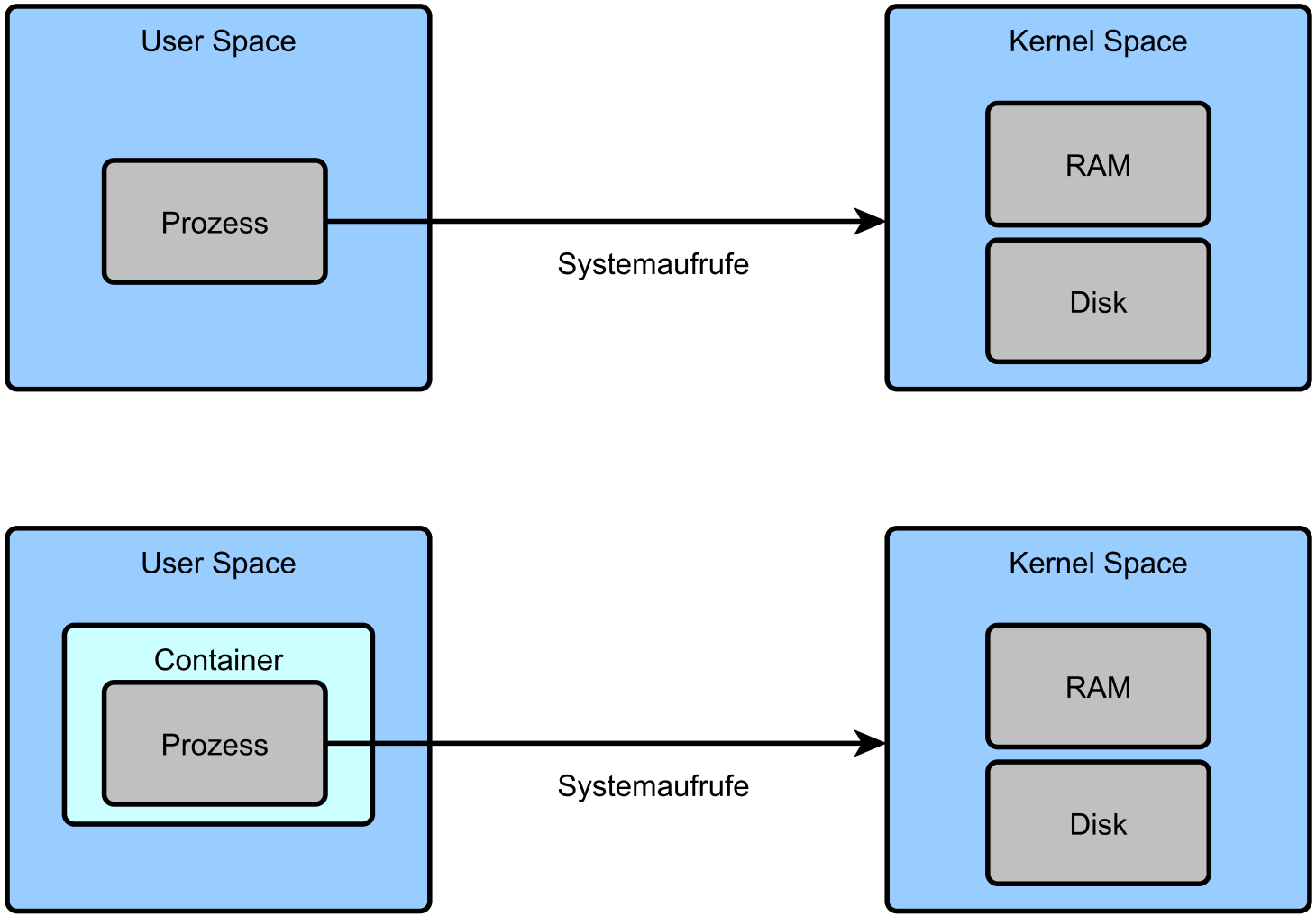


Container-Innenleben

Zwischen User Space und Kernel Space

- Alle Anwendungen des Betriebssystems inkl. der Container nutzen denselben Betriebssystem-Kern.
- Der Betriebssystem-Kern stellt diesen Anwendungen über Systemaufrufe ein API bereit.
- Die Version der API spielt eine wichtige Rolle, da sie der Kleber zwischen den Schichten ist.

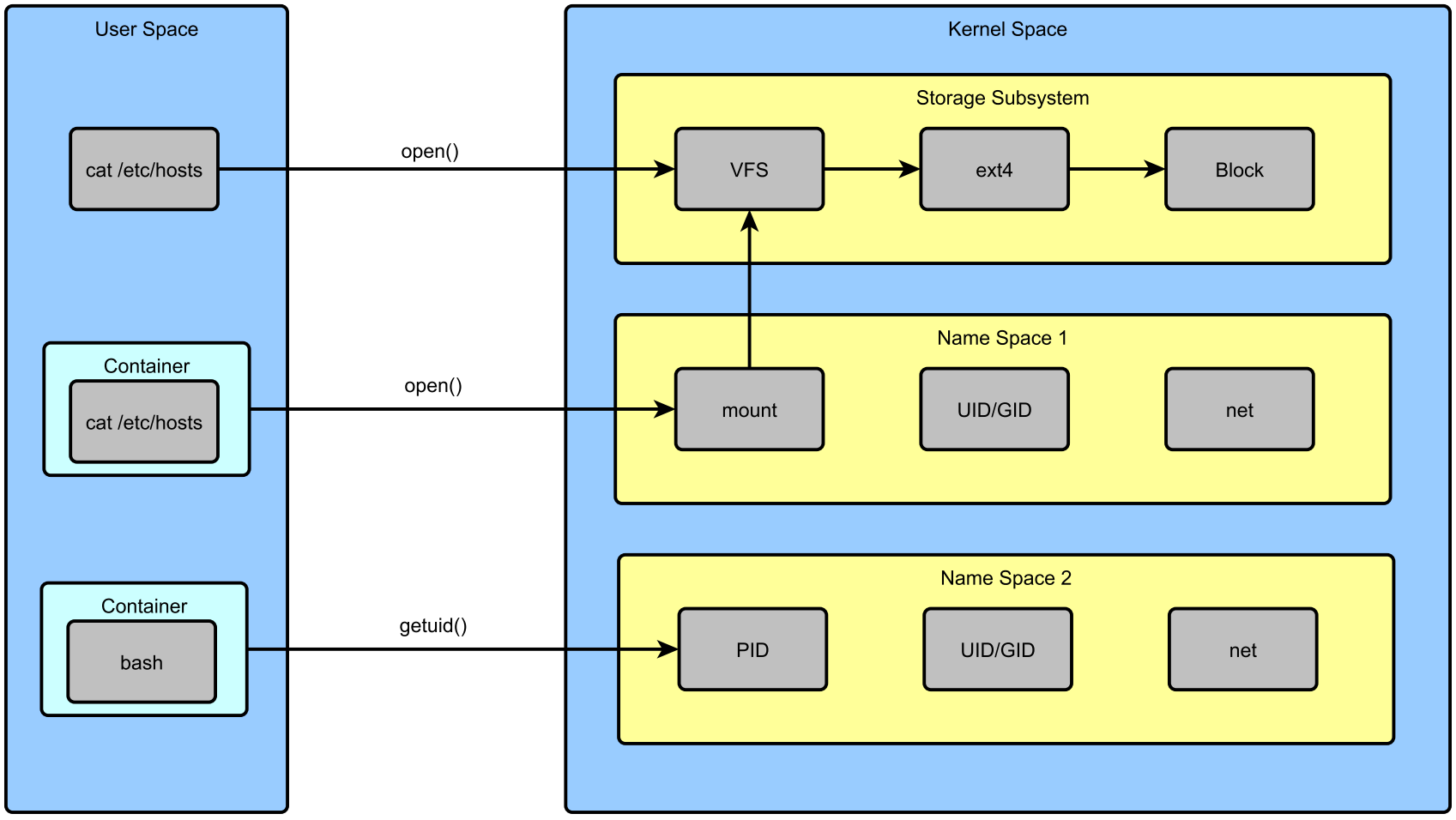
Prozess ruft Kern



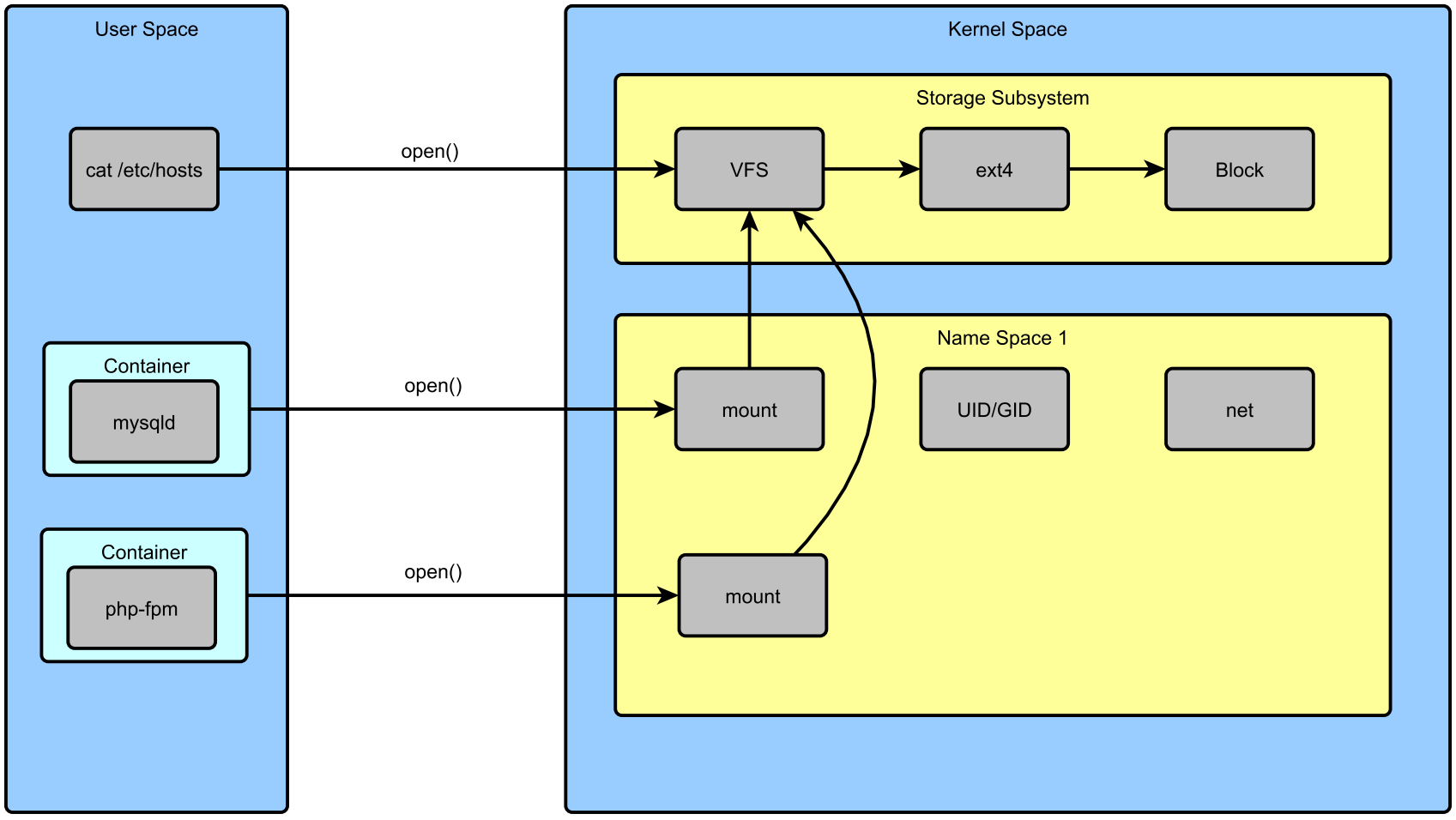
Systemaufrufe

- Welche Systemaufrufe ein Prozess kennt, bestimmt der Compiler bzw. Interpreter.
- Ein Betriebssystem-Kern versteht nur die in ihm implementierten Systemaufrufe.
- Passt der User Space nicht zum Kernel-API geht der Container über Bord.

Jedem Container sein Kernel Namespace



Container in einem Kernel Namespace



Möglichkeiten der Softwareverteilung

- Teilen von Quelltext/Binärdateien/Archiven
  - Beispiele: Git, RPM, APT, TAR, etc.
  - Vorteil: Es wird nur wenig Platz benötigt und dementsprechend ist die zu übertragende Datenmenge gering.
  - Nachteil: Abhängigkeiten wie z. B. Bibliotheken, Laufzeitumgebungen und Module werden häufig nicht mitgeliefert.
  - Hinweis: Softwarepaketierung ist nicht vergnügungssteuerpflichtig.
- Teilen in Form einer Virtuellen Maschine
  - Beispiele: OVA, OVF, VHD, VMDK, qcow2, etc.
  - Vorteil: Enhält alles, was man zum Ausführen der Anwendung braucht.
  - Nachteil: Die Formate sind groß und sperrig. VMs sind nicht optimal an Zielplattform angepasst. Sind meist Black Boxes.
- Teilen von Container-Repositories
  - Beispiele: registry.access.redhat.com, registry.redhat.io, quay.io, docker.io
  - Vorteil: Sind deutlich kleiner als VMs und enthalten alles was man zur Ausführung der Anwendung benötigt.
  - Nachteil: Auch hier gibt es das Black-Box-Problem.

Updateszenarien

- Container sind zustandslos.
- Der Inhalt eines Containers wird nicht aktualisiert.
- Der komplette Container wird durch eine neue Version ersetzt.

Daraus folgt:

- Persistente Datenspeicherung muss außerhalb des Containers erfolgen.
- Potenziell vereinfachte Update- und Rollback-Verfahren.
- Alte Herausforderungen bleiben bestehen (z. B. Anwendung ändert DB-Schema).

Mögliche Vorteile der Container

- Vorhandene Ressourcen können effizienter genutzt werden.
- Das Wunsch-Userland eines Herstellers (z. B. Ubuntu) kann auf dem Wunsch-Kern des BITS (RHEL) ausgeführt werden.
- Evtl. entgehen wir der Linux-Abhängigkeits-Hölle.

Mögliche Nachteile der Container

- Der Container-Inhalt veraltet und wird nie wieder aktualisiert. Es gibt keine Patches für Sicherheitslücken.
- Container entpuppen sich als trojanische Pferde, die nicht dokumentierte Workloads mitliefern.
- Es ist doch nur eine weitere Technologie, die betrieben werden muss.

Ende

- Vielen Dank für eure Aufmerksamkeit.
- Lesestoff findet ihr in meiner Linksammlung