Authentifizierung und Autorisierung in Kubernetes

Frühjahrsfachgespräch GUUG 2018

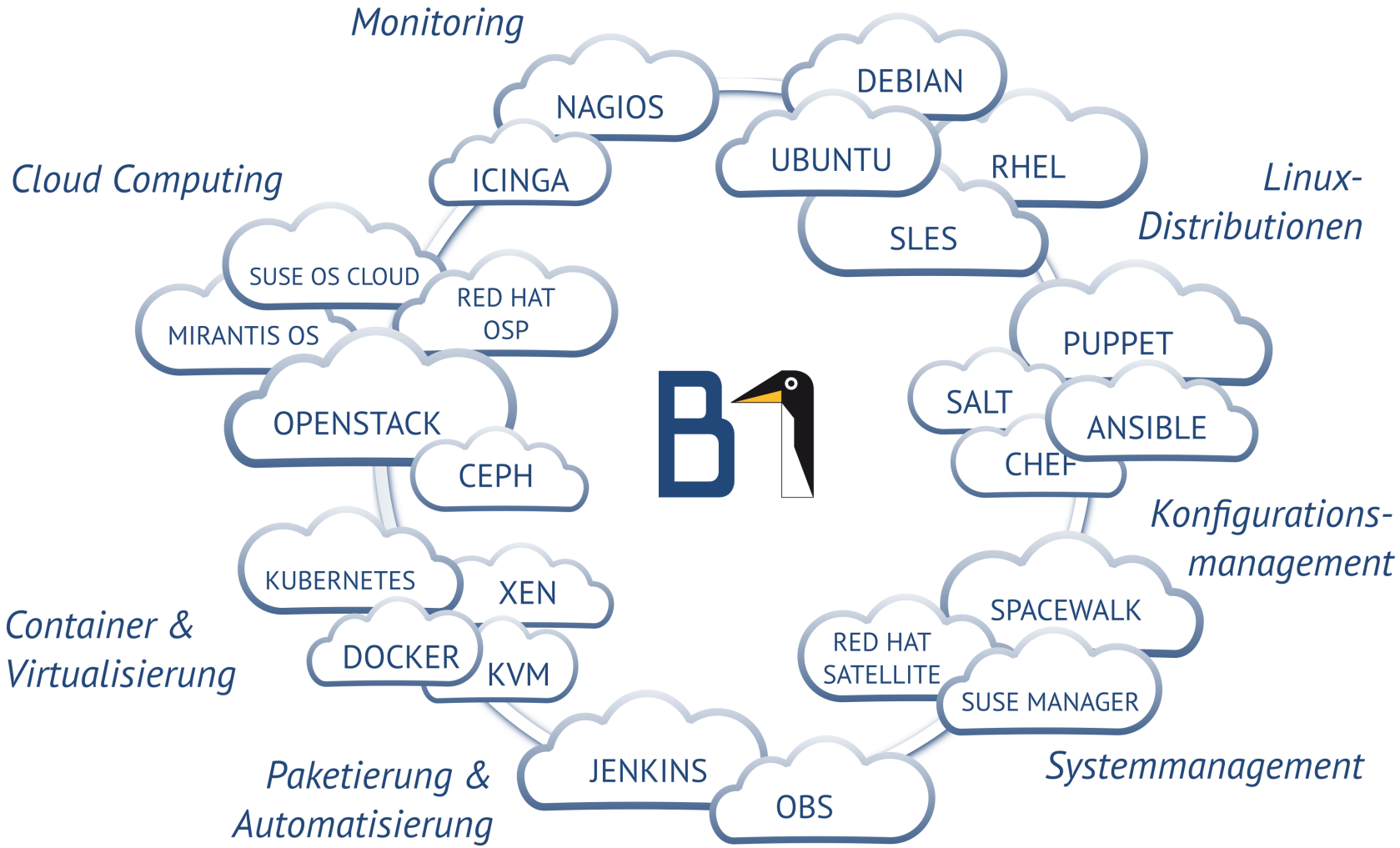
01. März 2018

Michael Steinfurth
Linux / Unix Consultant & Trainer
B1 Systems GmbH
steinfurth@b1-systems.de

Vorstellung B1 Systems

- gegründet 2004
- primär Linux/Open Source-Themen
- national & international tätig
- ca. 100 Mitarbeiter
- unabhängig von Soft- und Hardware-Herstellern
- Leistungsangebot:
  - Beratung & Consulting
  - Support
  - Entwicklung
  - Training
  - Betrieb
  - Lösungen
- Büros in Rockolding, Köln, Berlin & Dresden

Schwerpunkte

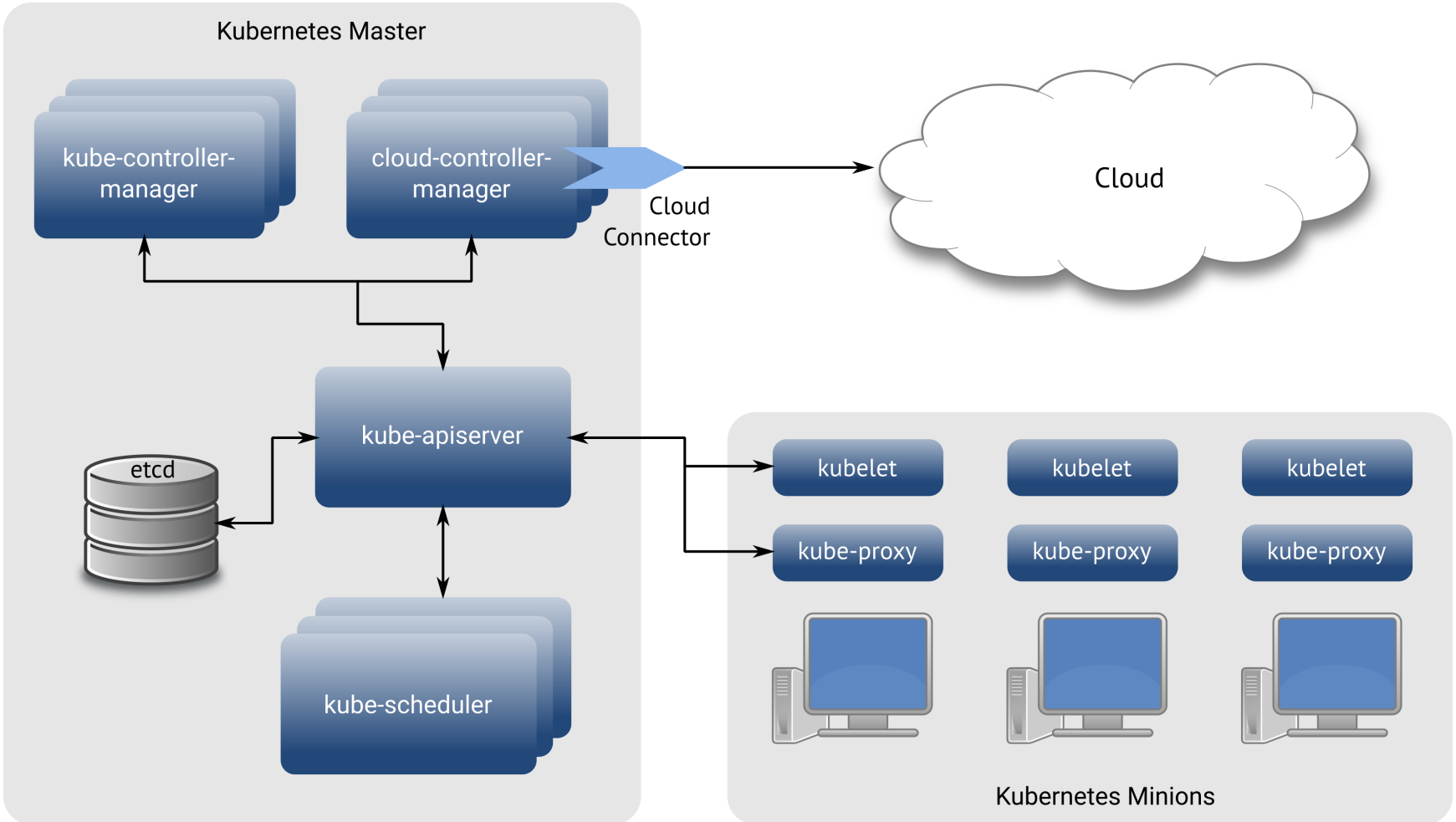


Einführung

Übersicht Aufbau Kubernetes

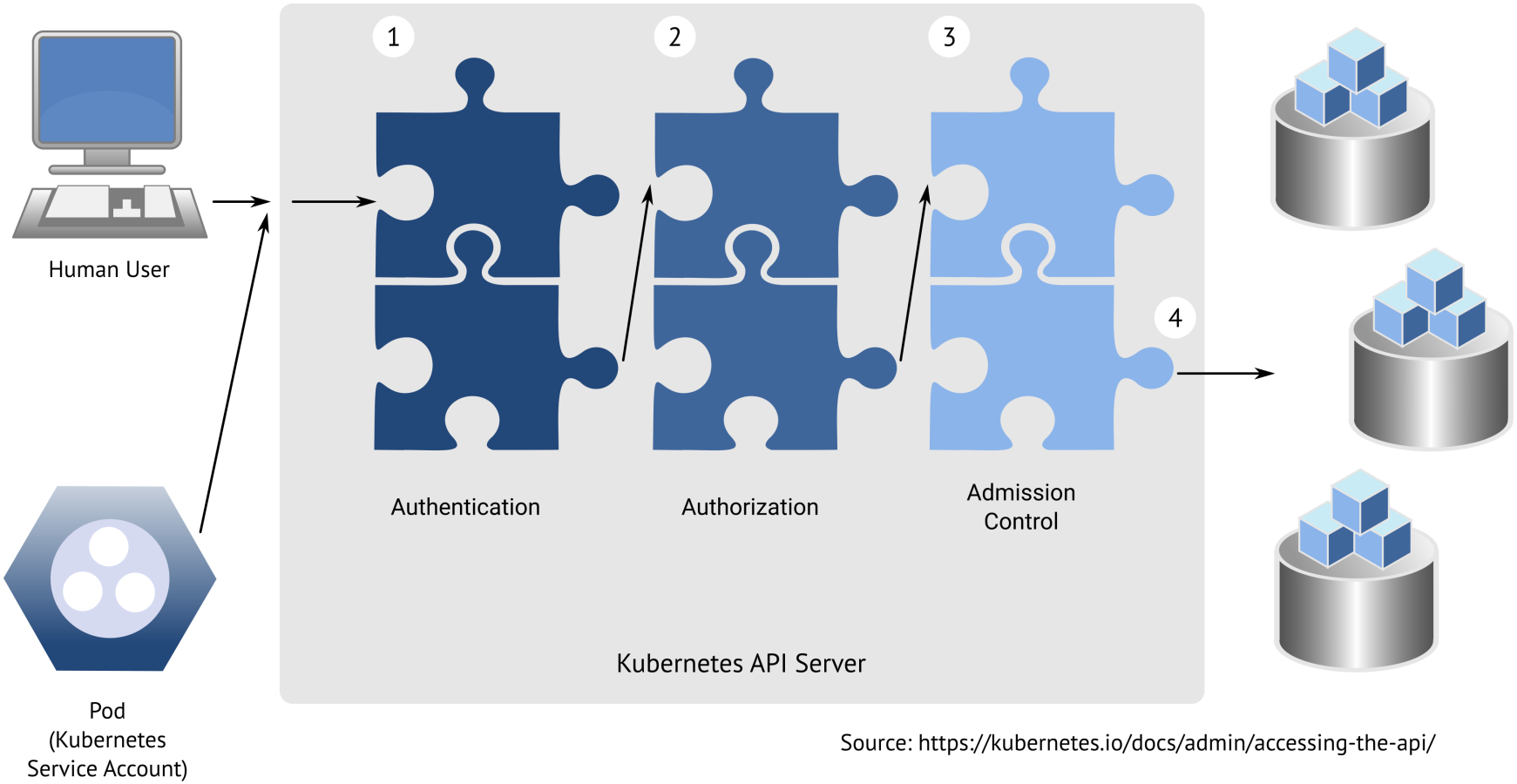
- Framework zum Verwalten von Containern auf mehreren Knoten
- hochverfügbare Applikationen in Containern
- Rolling Updates
- Kontrolle über Schnittstellen der Applikationen nach außen
- Lastverteilung mit automatische Skalierung

Architektur Kubernetes



Source: https://kubernetes.io/docs/concepts/architecture/cloud-controller/

Zugriff API



Source: https://kubernetes.io/docs/admin/accessing-the-api/

Anforderungen im Unternehmen

Authentifizierungsvorgaben

Typische Vorgaben in größeren Unternehmen:

- Anbindung an LDAP/AD
- Anbindung an ID-Services
- Anbindung an Cloud

Nutzer

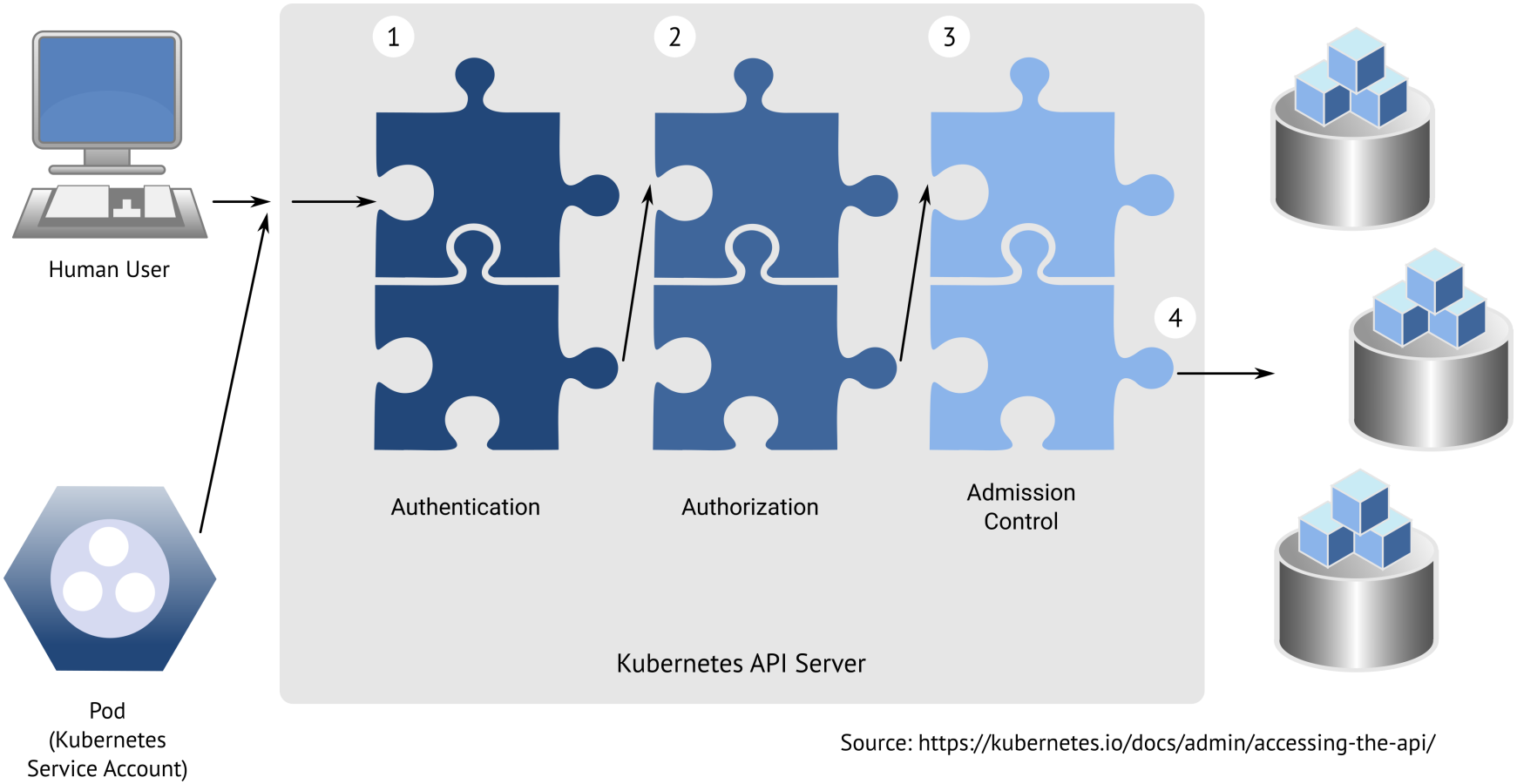
- Administratoren (Verwaltung)
- Entwickler (Appl. Container debuggen)
- Interne Abteilungen (z.B. Steuerung der Repliken)
- Externe Nutzer (Unternehmen tritt als Betreiber auf)

Direktes Beispiel

- Unternehmen mit mehreren Abteilungen
- Verantwortlicher Admin für den Kubernetes-Cluster
- Abteilungen mit eigenen Admins/Entwicklern

Authentifizierung

Zugriff API



Source: https://kubernetes.io/docs/admin/accessing-the-api/

API-Kommunikationen

- Nutzer
- kubelet, kubeproxy
- controller-manager, scheduler, cloud-controller

Authentifizierungsmethoden

- verschiedene Auth.-Methoden (mehrere konfigurierbar)
  - Zertifikate
  - Bearer Tokens
  - Authenticating Proxy
  - http auth
  - Spezial Service Accounts
- zusätzliche Attribute mitgeführt
  - Nutzername, UID, Gruppen, extra Felder
  - nicht genutzt von Authentifizierung
  - Authorisierungssystem verarbeitet sie

Zertifikate

- Zertifikate geprüft gegen eine mehre CA
- keine Revocation Liste
- Nutzer und Gruppe

```
/CN=bob/O=admin-namespace
```

- `--client-ca-file=` Option für api-server

Token-, Passwort-Datei

- Tokens oder statische Passwörter kommen aus Datei
- Datei und Tokens können nur durch Neustart geändert werden

```
token,nutzername,uid,"group1,group2,group3"
passwort,nutzername,uid,"group1,group2,group3"
```

- Startparameter:
  - `--token-auth-file=/pfadzudatei`
  - `--basic-auth-file=/pfadzudatei`

OpenID

- openID als Erweiterung von OAuth2
- vereinf. Ablauf
  1. Einloggen beim OpenID-Anbieter
  2. Rückgabe eines *access_token*, *id_token* and *refresh_token*
  3. kubectl nutzt die Tokens zum Authentifizerung und zur Aktualisierung des Tokens selbst
  4. API Server prüft die Gültigkeit mit dem Zertifikat vom OpenID-Anbieter
- Kubernetes API Server greift auf die Schnittstelle des OpenID-Anbieters zu
- `--oidc-`PARAMETER

Service Accounts

- automatische Zugänge für Anwendungen
- Service Accounts werden in pods eingehängt
- Zugriff aus dem Pod heraus auf API zugreifen
- Anwendungsfall: Ingress-Controller

Keystone

- Openstack Authentifizierer
- keine Gruppen
- Status experimental

```
--experimental-keystone-url=https://keystone.url:5000/v2.0/
--experimental-keystone-ca-file=/etc/kubernetes/auth/ca.pem
```

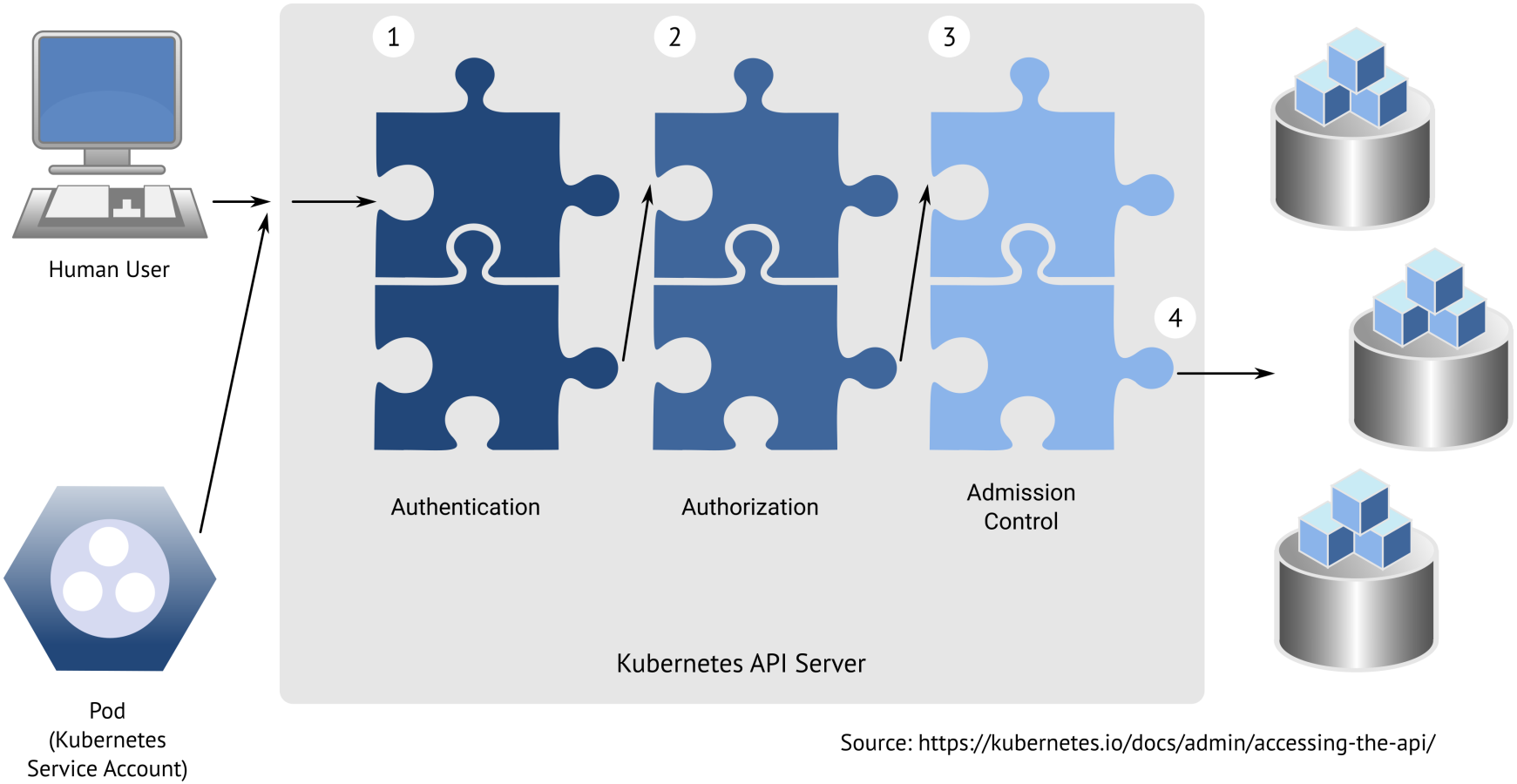
Was fehlt?

- kein natives AD / LDAP

↪ Lösung über Extra-Dienst OpenID wie Dex

Autorisierung

Zugriff API



Source: https://kubernetes.io/docs/admin/accessing-the-api/

Autorisierungskonzept 1/2

- direkt nach der gültigen(!) Authentifizierung setzt die Autorisierung ein
- Attribute (z.B. Nutzer, Gruppe, Ressource, Namespace) der Anfrage werden verarbeitet

↪ alle Attribute durch Richtlinien müssen erlaubt sein für ein positives Ergebnis

Autorisierungskonzept 2/2

- Autorisierung wird durch mehrere Module angeboten
- mehrere Module können gleichzeitig konfiguriert sein
  - werden sequentiell abgearbeitet
  - Zustimmung oder Ablehnung wird direkt umgesetzt
- RBAC, ABAC, Webhook, NodeAuthorization

Einführung RBAC

- Role-Based Access Control
- erlaubt Admins dynamische Veränderungen zur Laufzeit
- *Role* und *ClusterRole*
- *RoleBinding* und *ClusterRoleBinding*

(Cluster-)Rollen

- bestehen aus Regeln, die Permissions beinhalten
- nur Erlaubt-Regeln, keine Verboten-Regeln
- Role gilt innerhalb eines Namespaces
- ClusterRole gilt clusterweit (z.B. *Nodes*-Ressource, *–all-namespaces*)

ClusterRole View

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: view
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - persistentvolumeclaims
  - pods
  - services
  verbs:
  - get
  - list
  - watch
```

(Cluster-)RoleBinding

- Zuordnung der Role zu einem Nutzer oder einer Gruppe
- Gültigkeitsbestimmung
  - RoleBinding → pro Namespace
  - ClusterRoleBinding → clusterweit

↪ einmalig definierte ClusterRole kann an unterschiedliche Namespaces gebunden werden (RoleBinding)

Beispiel Umsetzung

- Unternehmen mit mehreren Abteilungen
- Verantwortlicher Admin für den Kubernetes-Cluster
- Abteilungen mit eigenen Admins/Entwicklern

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096