Kubernetes
Multi-Tenancy & User-Management

By Fabian Kramm

@FabianKramm

DevSpace

Hi! I am Fabian.

- CTO @ DevSpace Technologies Inc.
- We help companies to provision secure Kubernetes sandbox environments
- Open-Source Contributor
- Prev. worked at SAP & sovanta AG
- Trivia: Born in Mannheim

@FabianKramm

What is Multi-Tenancy?

Source: platform9.com

Why Multi-Tenancy?

Separate Clusters vs. Cluster Sharing

- **Higher Compute Cost**
  - Redundant cluster-wide resources
    (e.g. control planes, operators, storage and networking services)
  - Less efficient cluster auto-scaling
- **Higher Maintenance Cost**
  - Number of clusters explodes with adoption spreading across teams and applications
  - Hard to keep everything in sync

Separate Clusters vs. Cluster Sharing

- **Lower Compute Cost**
  - Shared cluster-wide resources
  - Less efficient cluster auto-scaling
- **Lower Maintenance Cost**
  - Reduce number of clusters (sharing if it makes sense)

Hard Tenancy vs Soft Tenancy

- **Hard Tenancy:**
  - No trust between tenants
  - Multiple users from multiple different places are sharing the cluster

  => Lots of configuration necessary

- **Soft Tenancy:**
  - Tenants usually part of the same organization
  - *Small* chance of malicious behaviour
  - “prevent accidents” rather than assuming the worst

  => Rather easy to achieve

Overview

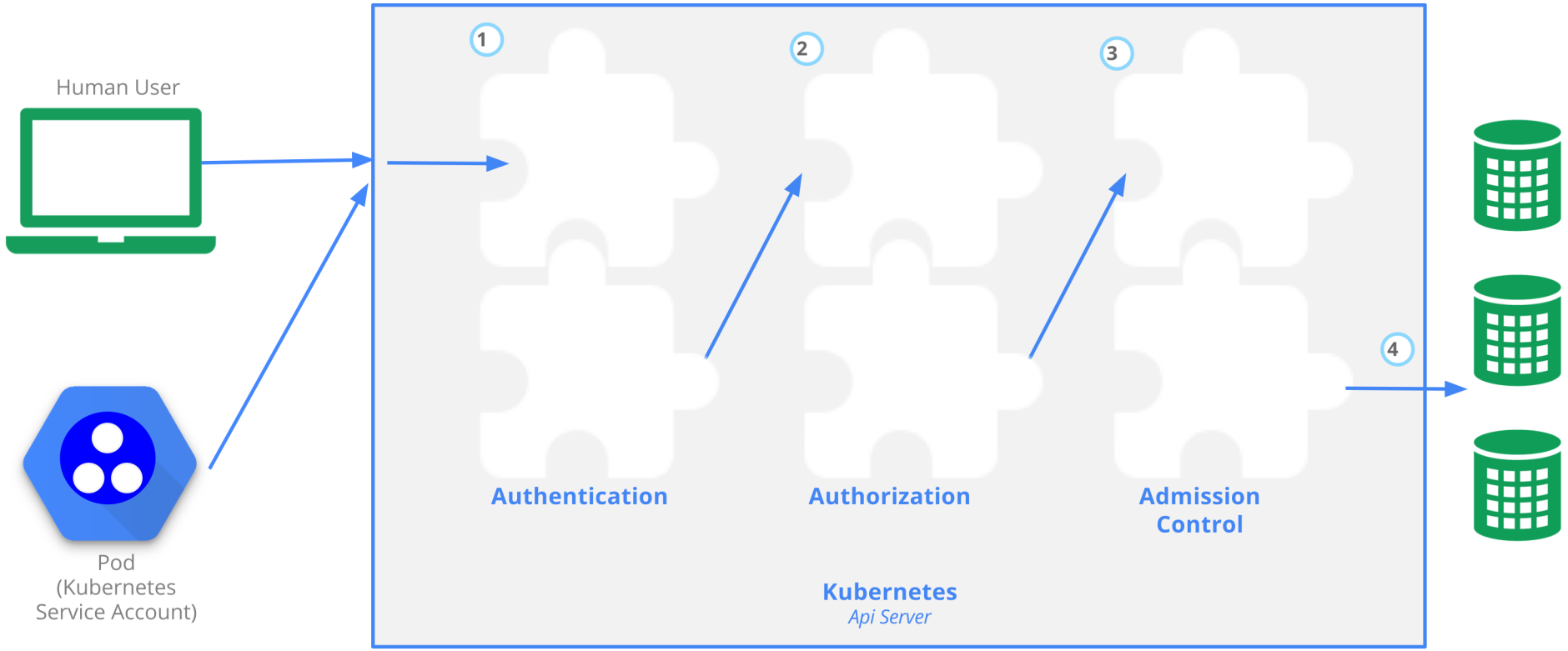
- **Basics**
  - Access Control
  - Namespaces
- **Authentication**
  - OpenID Connect
  - Service Accounts
  - Rancher
- **Authorization**
- **Admission Control**
  - Resource Quotas
  - Limit Ranges
  - PodSecurityPolicies
  - Open Policy Agent
- **Advanced Isolation**
  - Network Policies
  - Sandboxing Containers

Namespaces

- **Namespace = Scope for names of Kubernetes resources**
- **Some resources are cluster-wide** (Persistent Volume)
- **Cannot be nested** (yet?)

- **Reserved namespaces:**
  - kube-system (control plane)
  - kube-public (currently used for kubeadm)
  - default

Access Control in Kubernetes

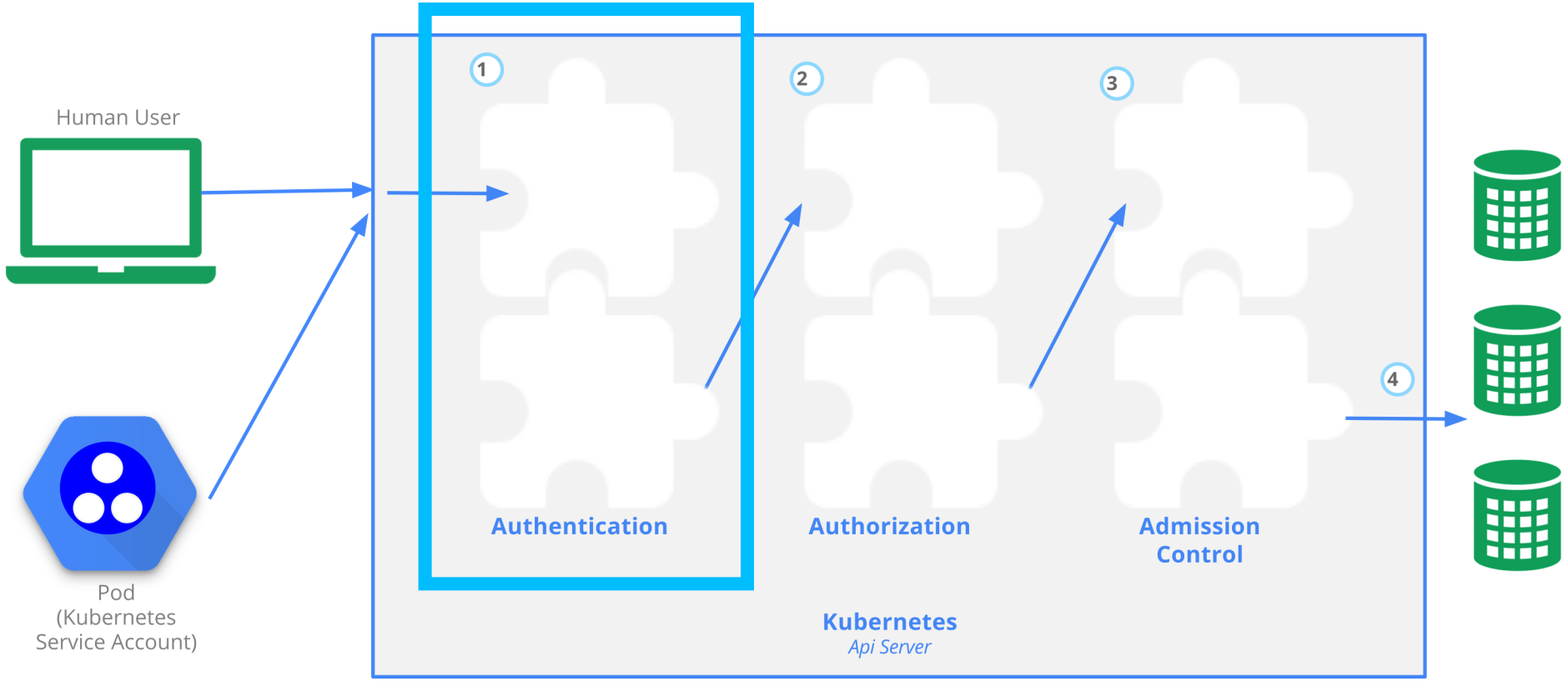


Source: Kubernetes Documentation

Overview

- **Basics**
  - Access Control
  - Namespaces
- **Authentication**
  - OpenID Connect
  - Service Accounts
  - Rancher
- **Authorization**
- **Admission Control**
  - Resource Quotas
  - Limit Ranges
  - PodSecurityPolicies
  - Open Policy Agent
- **Advanced Isolation**
  - Network Policies
  - Sandboxing Containers

Where are we?



Source: Kubernetes Documentation

User Authentication

- **Several Methods**
  - Client Certificates (e.g. with flag --client-ca-file=CA-FILE.pem)
  - HTTP Basic Auth (e.g. with flag --basic-auth-file=USER-FILE.csv)
  - Bearer Tokens
    - using static token file (with flag --token-auth-file=TOKEN-FILE.csv)
    - using OIDC (OAuth2) => Dex project
    - using token verification via webhook
    - using service account tokens
  - Authentication Proxy

⇒ **Requires Kubernetes api server access**

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3M
iOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiw
ia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9
uYW1lc3BhY2UiOiJjYyIsImt1YmVybmV0ZXMuaW8
vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJ
kZWZhdWx0LXRva2VuLTJoa2ZkIiwia3ViZXJuZXR
lcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWF
jY291bnQubmFtZSI6ImRlZmF1bHQiLCJrdWJlcm5
ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2U
tYWNjb3VudC51aWQiOiI2MGQyMTg2MS0wNjQwLTE
xZWEtOGY1ZC0wMjUwMDAwMDAwMDEiLCJzdWIiOiJ
zeXN0ZW06c2VydmljZWFjY291bnQ6Y2M6ZGVmYXV
sdCJ9.rDmdpTKruM1KRB3Y3ygxSO-
7SMBzQ08wQBaSWJ7-
K4mTyekUKBvyLvkOqfkUdHwAljntk_CeuMfcBnC6
Aed1j8PHwYBr8MJWpL376V2FEQPB90YJWYcH98bW
AL_kw_QHuaERU8inc0R9cqvg5jjef-
won6VpsLWnUeOLfhQo7sNm_VIcunnUH0k6VUeR3a
Etxx7yPuGP2DT_a0eeUGBGAc94QWdQxssu38pxhY
5Pe8nyaH28e2kHcE0fTck0z82nHBseJDwEmfWTMX
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "kid": ""
}
```

PAYLOAD: DATA

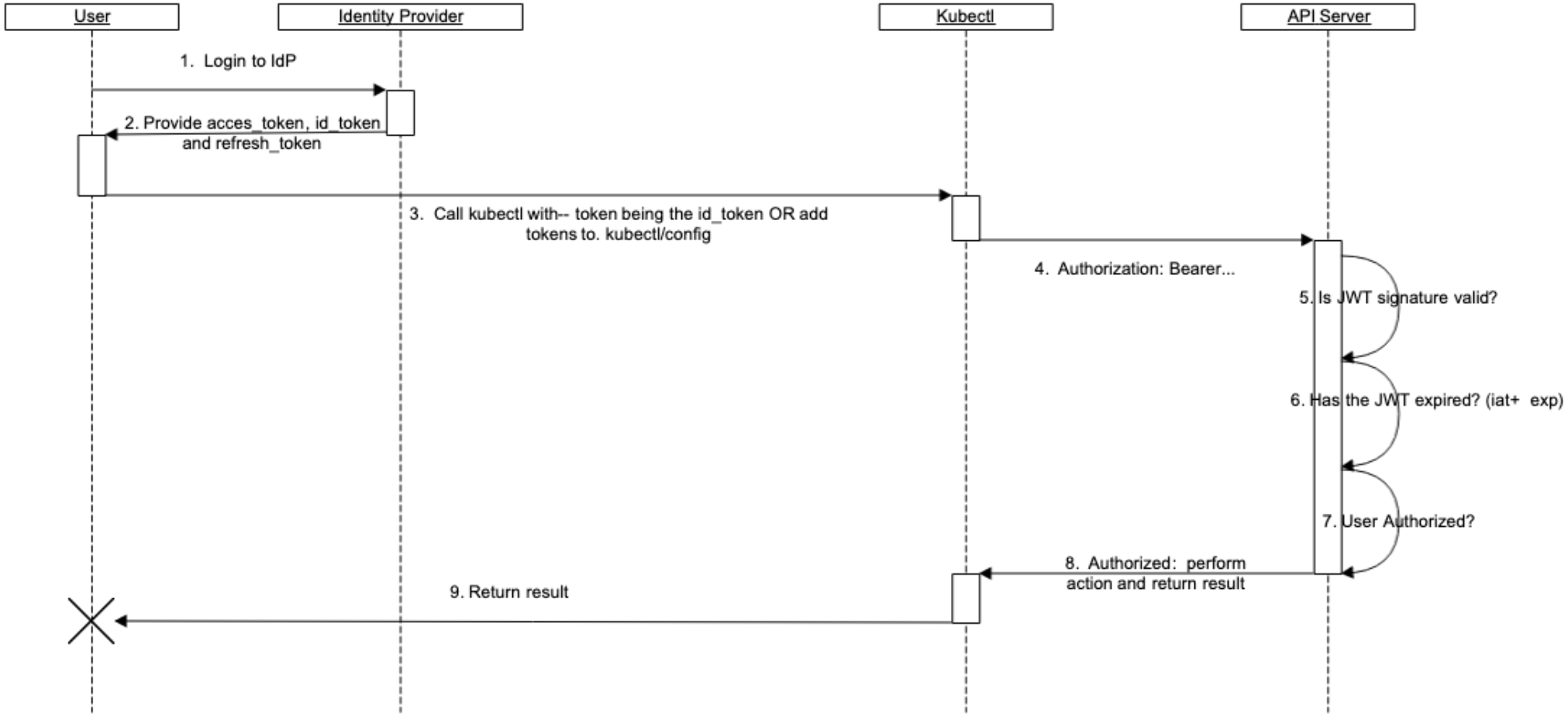
```
{
  "iss": "kubernetes/serviceaccount",
  "kubernetes.io/serviceaccount/namespace": "cc",
  "kubernetes.io/serviceaccount/secret.name": "default-
token-2hkfd",
  "kubernetes.io/serviceaccount/service-account.name":
"default",
  "kubernetes.io/serviceaccount/service-account.uid":
"60d21861-0640-11ea-8f5d-025000000001",
  "sub": "system:serviceaccount:cc:default"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key or Certificate. Ent
```

OpenID Connect

Source: Kubernetes Documentation



ServiceAccounts

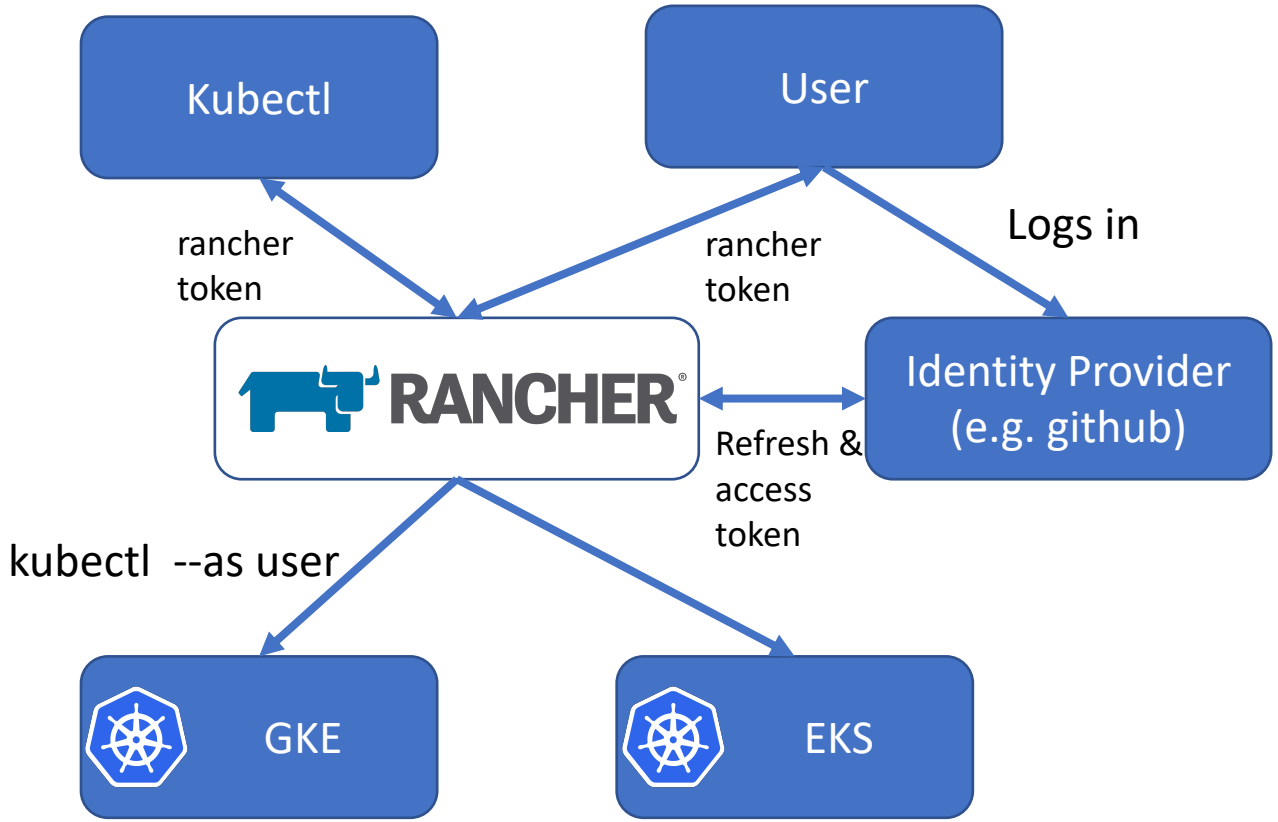
- **Usually used by in cluster services**
- **Kubernetes auto-generates ServiceAccount “default”**
- **Every container mounts a volume with the token of the ServiceAccount referenced by the container’s pod (can be disabled)**
  /var/run/secrets/kubernetes.io/serviceaccount

- **Only Kubernetes managed account resource**
  (Kubernetes does not know a user resource)

```yaml
apiVersion: v1
kind: Config
clusters:
  - cluster:
      certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSU
      server: https://kubernetes.docker.internal:6443
    name: docker
contexts:
  - context:
      cluster: docker
      user: docker
    name: docker
current-context: docker
users:
  - name: docker
    user:
      token: eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3Nlcn
```

Rancher Authentication Proxy

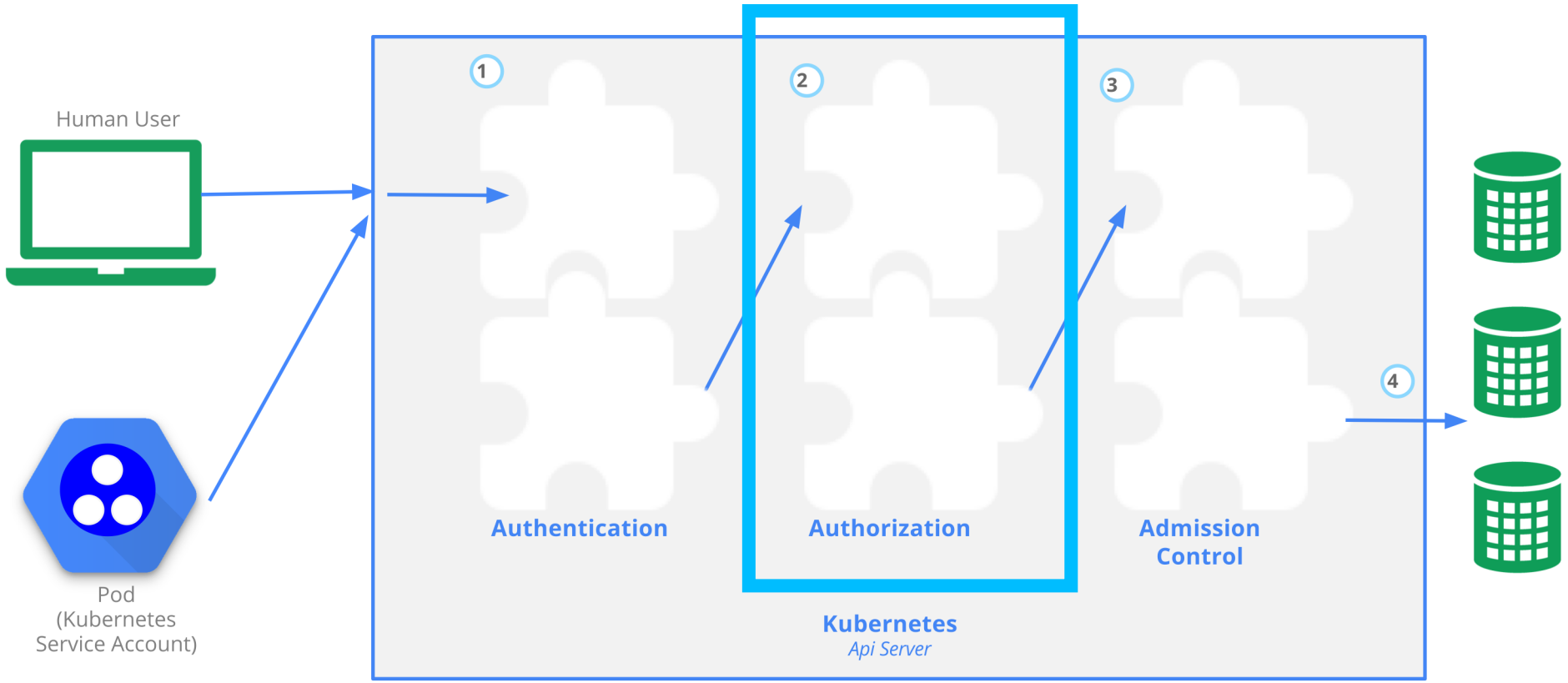
- Rancher forwards user requests to Kubernetes using a service account and impersonation
- Supports local or external authentication (e.g. github, ldap etc.)
- Users can access multiple clusters with one single set of credentials



Overview

- **Basics**
  - Access Control
  - Namespaces
- **Authentication**
  - OpenID Connect
  - Service Accounts
  - Rancher
- **Authorization**
- **Admission Control**
  - Resource Quotas
  - Limit Ranges
  - PodSecurityPolicies
  - Open Policy Agent
- **Advanced Isolation**
  - Network Policies
  - Sandboxing Containers

Where are we?



Source: Kubernetes Documentation

RBAC (Role-based access control)

- **Stable since Kubernetes 1.8**
- **Successor of ABAC**
  (using kubectl, via Kubernetes resources)

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role # Namespace scoped
metadata:
  namespace: default
  name: pod-reader
rules:
  - apiGroups: [""] # "" indicates the core API group
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
```

- **Roles define what type of access is allowed on resources**
  - Roles for namespace scoped authorization
  - ClusterRoles for cluster scoped authorization

RBAC – RoleBindings

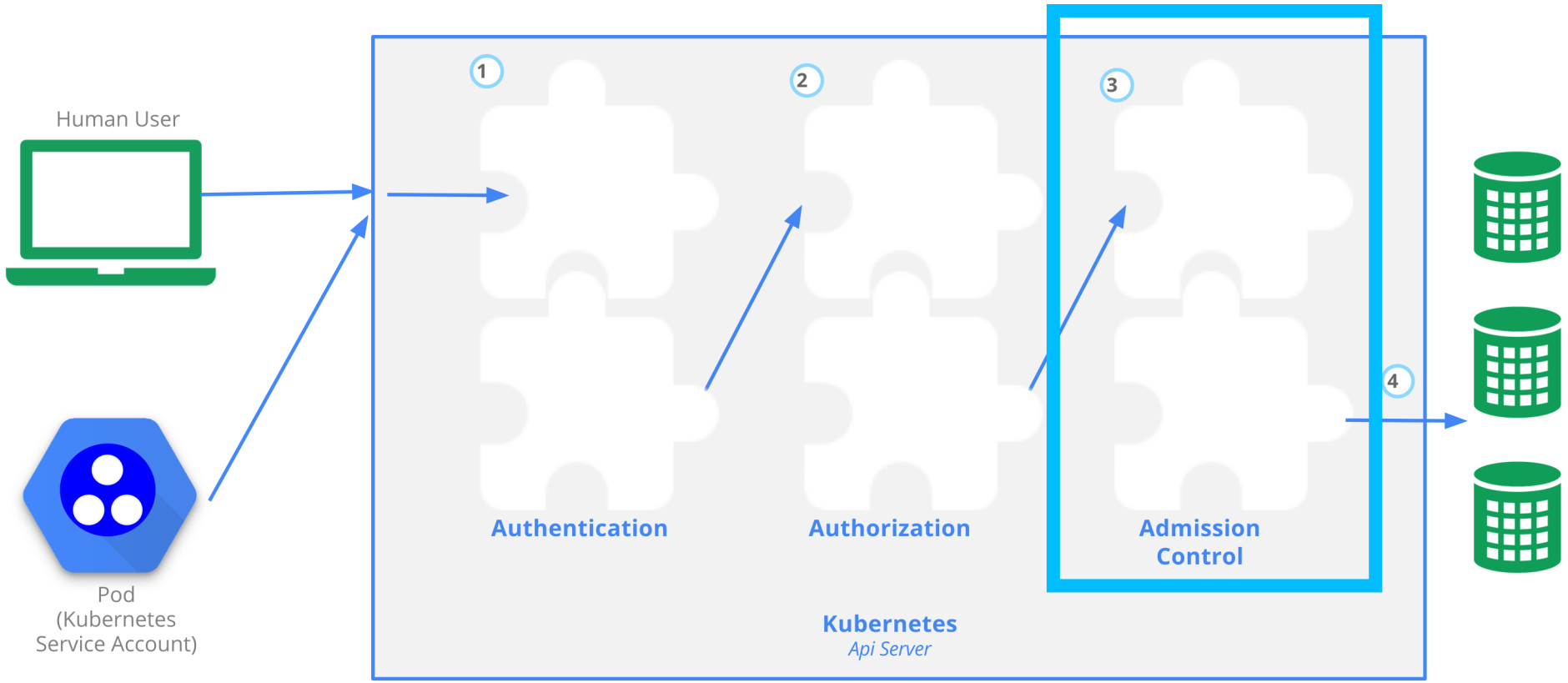
- **Role bindings assign Users / Groups / ServiceAccounts to roles**
  - ClusterRoleBinding binds ClusterRoles
  - RoleBinding binds ClusterRoles / Roles
- **Multiple subjects per binding possible**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "fabian" to
# read pods in the "default" namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: fabian
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # Role or ClusterRole
  kind: Role
  # this must match the name of the Role
  # or ClusterRole you wish to bind to
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Overview

- **Basics**
  - Access Control
  - Namespaces
- **Authentication**
  - OpenID Connect
  - Service Accounts
  - Rancher
- **Authorization**
- **Admission Control**
  - Resource Quotas
  - Limit Ranges
  - PodSecurityPolicies
  - Open Policy Agent
- **Advanced Isolation**
  - Network Policies
  - Sandboxing Containers

Where are we?

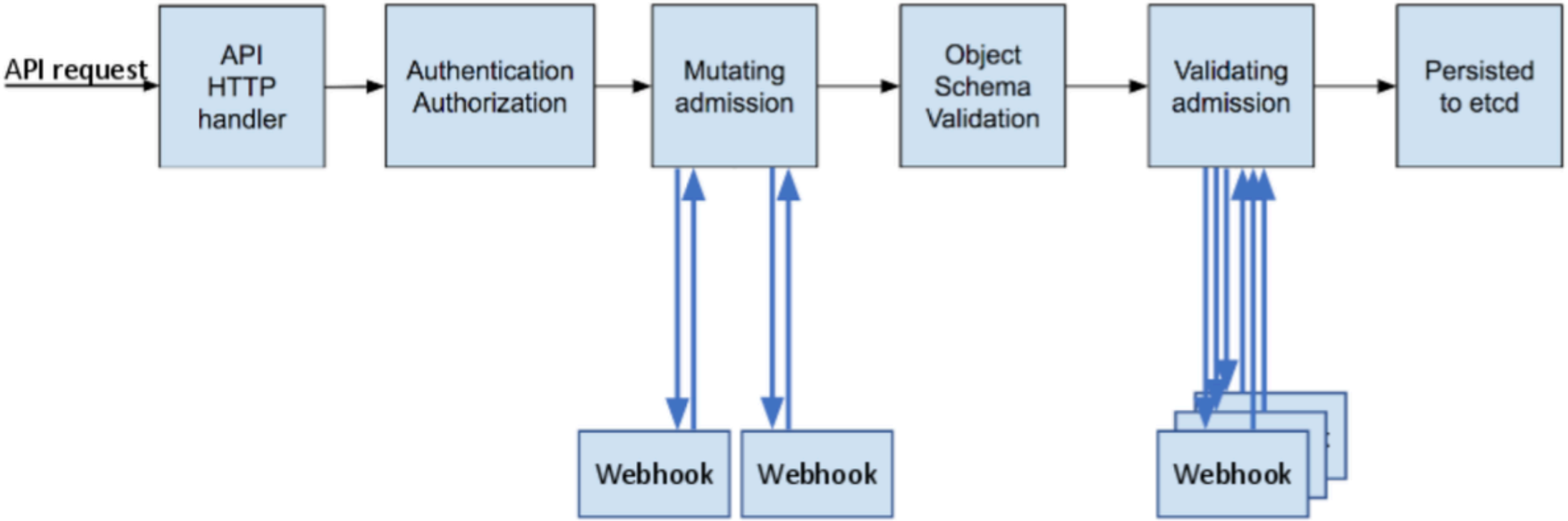


Source: Kubernetes Documentation

Admission Controllers

- **Intercept requests to the API server**
- **Types:**
  - Mutating – Change the resources sent within the request (called first)
  - Validating – Allow or deny the resources sent within the request
- **30+ prebuilt admission controllers available**
- MutatingAdmissionWebhook **and** ValidatingAdmissionWebhook **allow to add dynamic admission control via webhooks**

Admission Controllers



Source: Kubernetes Blog

ResourceQuota

- **Enforced as admission controller**
- **Limits aggregate resource consumption per namespace**
- **Common cases:**
  - Limit total cpu/memory requests/limits
  - Limit total persistent storage or ephemeral storage
  - Limit total number of pods/services/secrets/configmaps/...
  - Limit total number of resource quotas ;-)

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: myspace
spec:
  hard:
    count/roles.rbac.authorization.k8s.io: "5"
    pods: "100"
    requests.cpu: "8"
    limits.cpu: "16"
```

LimitRange

- Allows to set min | max | default values for cpu | memory | storage
- Prevents that a single pod/container consumes all available resources in a namespace (Resource Quota)
- Also available for persistent volume claims

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: mylimitrange
  namespace: myspace
spec:
  # Limits the container
  # resources to 1-4 cores
  limits:
    - max:
        cpu: "4"
      min:
        cpu: "1"
      type: Container
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      resources:
        requests:
          cpu: "0"
          memory: "0"
        limits:
          cpu: "0"
          memory: "0"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: /mnt
          name: bad-volume
  volumes:
    - name: bad-volume
      hostPath:
        path: /
```

PodSecurityPolicy

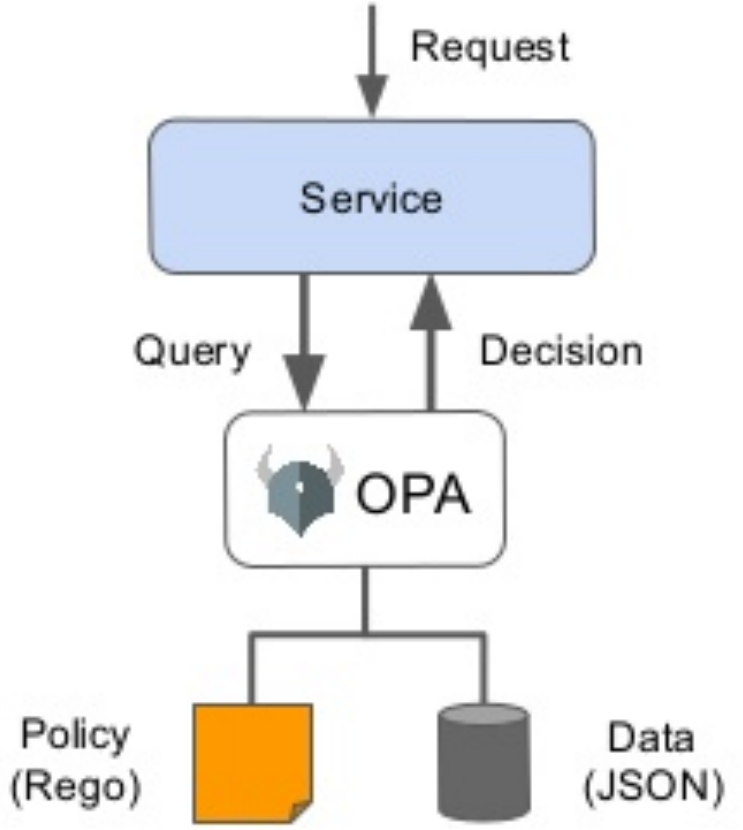
- **Allows admins to limit the capabilities of pods**
- **Once enabled no pods can be deployed without active policy**
- **Common cases:**
  - Disable privileged containers
  - Disable use of hostNetwork
  - Disable certain volume types (e.g. hostPath)
  - Restrict use of Linux capabilities (e.g. CAP_CHOWN)
- **Assignable to roles**

```yaml
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: policy
spec:
  allowPrivilegeEscalation: true
  fsGroup:
    rule: RunAsAny
  hostPorts:
  - max: 0
    min: 0
  runAsUser:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - persistentVolumeClaim
```

Open Policy Agent (OPA)

- Dynamic admission controller
- Language for policy logic: Rego
- Allows to define admission rules as Kubernetes resources
- Audits

- Gatekeeper project adds custom resource definitions for policies



Source: OPA Documentation

Overview

- **Basics**
  - Access Control
  - Namespaces
- **Authentication**
  - OpenID Connect
  - Service Accounts
  - Rancher
- **Authorization**
- **Admission Control**
  - Resource Quotas
  - Limit Ranges
  - PodSecurityPolicies
  - Open Policy Agent
- **Advanced Isolation**
  - Network Policies
  - Sandboxing Containers

NetworkPolicy

- Restricts the incoming (ingress) and outgoing (egress) network traffic of pods
- Pods initially allow all ingress and egress traffic if they are not selected by a network policy
- Must be supported by the underlying network plugin!

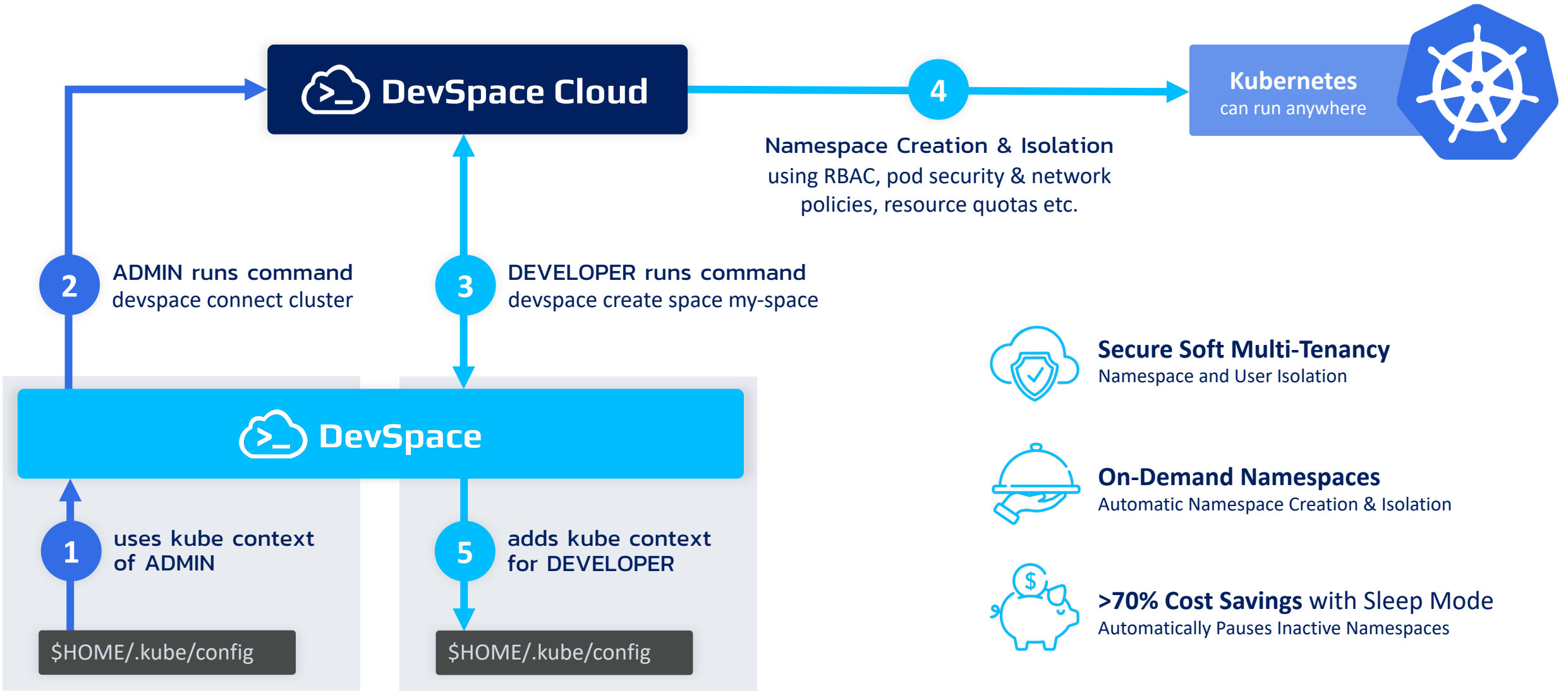
```yaml
apiVersion: extensions/v1beta1
kind: NetworkPolicy
metadata:
  name: networkpolicy
  namespace: myspace
spec:
  egress:
  - to:
    - podSelector: {}
    - namespaceSelector:
        matchLabels:
          my.cloud.com/traffic-allowed: allowed
  podSelector: {}
  policyTypes:
  - Egress
```

Sandboxing Containers

- **Containers != VMs**
  - Sharing a kernel bears risks
  - Vulnerabilities can lead to programs escaping the “soft” boundaries of a container

- **Sandboxing Methods**
  - Container Runtimes for Isolation
    - gVisor (user-space kernel)
    - Katacontainers (lightweight vm)
  - Nodeless Kubernetes
    - Virtual Kubelet
    - Elotl

How are we using all of these features in DevSpace Cloud?



Secure Soft Multi-Tenancy
Namespace and User Isolation

On-Demand Namespaces
Automatic Namespace Creation & Isolation

>70% Cost Savings with Sleep Mode
Automatically Pauses Inactive Namespaces

I'm here to help. Get in touch.

- Connect on Twitter: @FabianKramm
- Email me: fk@devspace.cloud
- Check out our projects on GitHub:

DevSpace

Get the fastest development workflow for Kubernetes

DevSpace on GitHub

DevSpace Cloud

Turn any Kubernetes cluster into a powerful development platform

DevSpace Cloud on GitHub