kublr

Kubernetes RBAC 101

Oleg Chunikhin | CTO, Kublr

Introductions

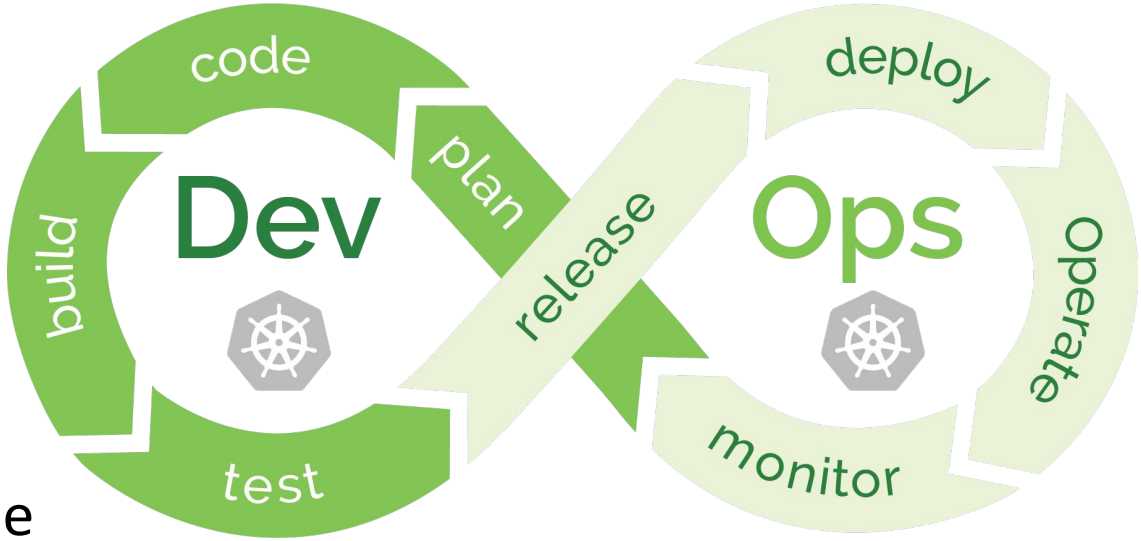
Oleg Chunikhin
CTO, Kublr

- **20 years** in software architecture & development
- Working w/ Kubernetes **since its release** in 2015
- **Software architect behind Kublr**—an enterprise ready container management platform
- Twitter **@olgch**

Enterprise Kubernetes Needs

Developers

- Self-service
- Compatible
- Conformant
- Configurable
- Open & Flexible



- Security
- Reliability
- Performance
- Portability

SRE/Ops/DevOps/SecOps

- Governance
- Org multi-tenancy
- Single pane of glass
- Operations
- Monitoring
- Log collection
- Image management
- Identity management

OPERATIONS

SECURITY & GOVERNANCE

Automation

Infrastructure

Logging

Monitoring

Observability

Custom Clusters

API

Usage Reporting

Ingress

Storage

Networking

Container Registry

CI / CD

App Mgmt

Container Runtime

Kubernetes

Infrastructure

RBAC

IAM

Air Gap

TLS

Certificate Rotation

Audit

@olgch; @kublr

Kubernetes Access Control

- Who can do what with which resource
- Authentication
- Authorization
- RBAC
- Use-cases and gotchas

@olgch; @kublr

Who Can Do What in a Cluster

Three groups

Subjects

User 

Group 

Service account 

Operations

list
get
create
update
delete
watch
patch

get
post
put
delete

Resources

Pods
Nodes
ConfigMaps
Secrets
Deployments
…

Connected through access control

Kubernetes API Request Attributes

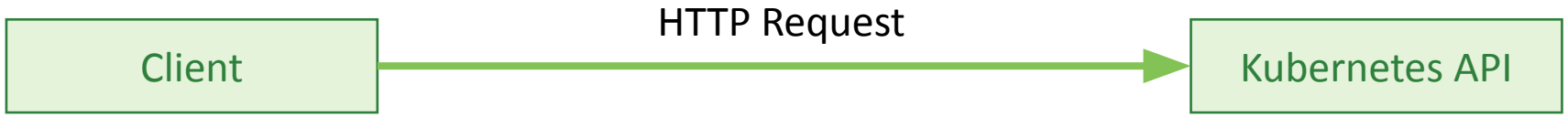
- **Authentication**
  - **User** – the user string
  - **Group** – the list of group names
  - **Extra** – a map of arbitrary keys
- **API** – non-resource or API resource flag
- **API resource request**
  - **API request verb** – lowercased resource verb
  - **Namespace** – the namespace
  - **API group** - The API Group being accessed
  - **Resource** – the resource ID
  - **Subresource** – the sub-resource
- **Non-resource request**
  - **HTTP request verb** – lowercased HTTP method
  - **Request path** – non-resource request path.

Verbs

- **Common API resource request** :
  get, list, watch,
  create, update, patch,
  delete, deletecollection
- **Special API resource request**:
  use (PodSecurityPolicy),
  bind, escalate (Role, ClusterRole),
  impersonate (User, Group, SA),
  userextras
- **HTTP request** verbs:
  get, head, post, put,
  patch, delete

Kubernetes API Client, Tools

- Any HTTP client
- `curl` – good for experiments
- `kubectl` – Kubernetes CLI



- `jq` – honorable mention – JSON visualization and processing

Kubernetes API Client, Example

- **curl**

```
curl -k -v -XGET -H 'Authorization: Bearer ***' \
  'https://52.44.121.181:443/api/v1/nodes?limit=50' | jq -C . | less -R
```

- **kubectl**

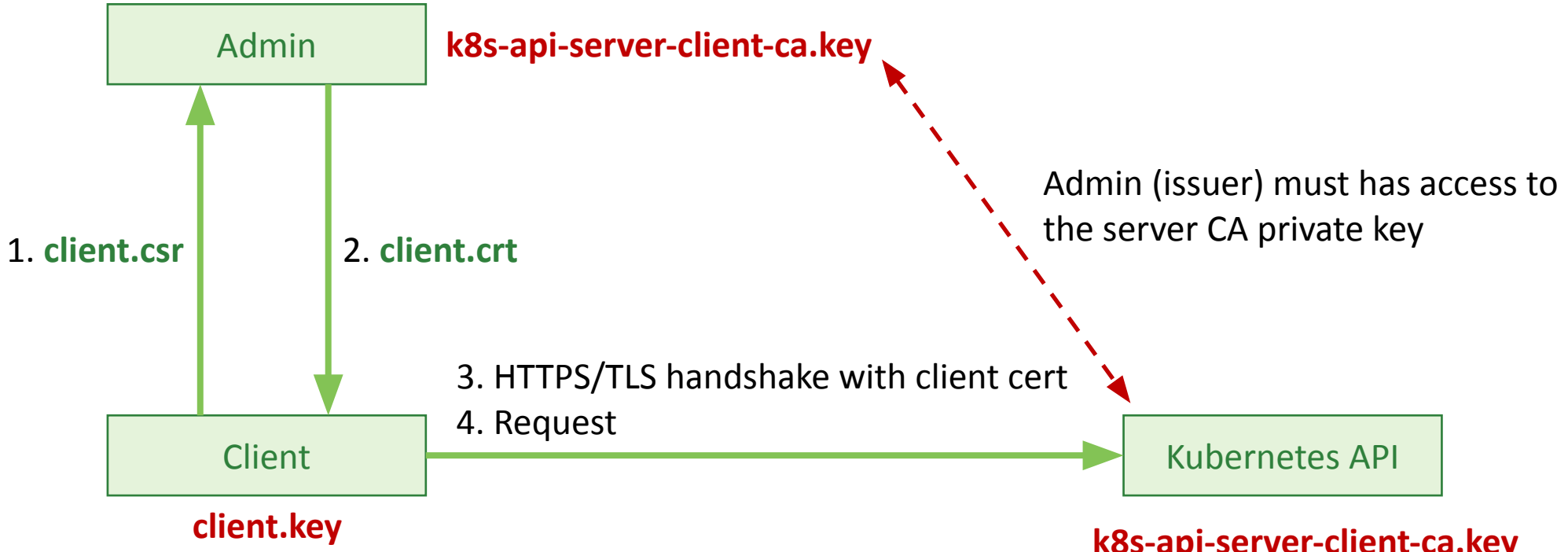
```
kubectl --kubeconfig=kc.yaml get nodes

export KUBECONFIG="$(pwd)/config.yaml"
kubectl get nodes

kubectl get nodes --v=9
```

```
apiVersion: v1
kind: Config
clusters:
- name: demo-rbac
  cluster:
    certificate-authority-data: ***
    server: https://52.44.121.181:443
users:
- name: demo-rbac-admin-token
  user:
    token: ***
contexts:
- name: demo-rbac
  context:
    cluster: demo-rbac
    user: demo-rbac-admin-token
current-context: demo-rbac
```

Authentication: Mechanisms

Mechanism	Secret Source	Usage
X509 Client Certs	CSR generated externally and signed with the cluster CA key	Enterprise CA / PKI
	Via Kubernetes API CertificateSigningRequest	Kubernetes cluster admin
Bearer token	Bootstrap token	Internal use
	Node authentication token	Internal use
	Static token file	Insecure
	ServiceAccount token	Pods, containers, applications, *users*
	OIDC token	Users
HTTP Basic auth	Static password file	Insecure
Auth proxy	N/A (trust proxy)	Integration
Impersonate	N/A (trust account)	Integration and administration

Authentication: X509 Client Cert, PKI




```
--client-ca-file=k8s-api-server-client-ca.key
```

Authentication: X509 Client Cert, PKI Example

- **User:** generate user private key (if not exist)

```
openssl genrsa -out user1.key 2048
```

- **User:** generate user CSR

```
openssl req -new -key user1.key -out user1.csr -subj "/CN=user1/O=group1/O=group2"
```

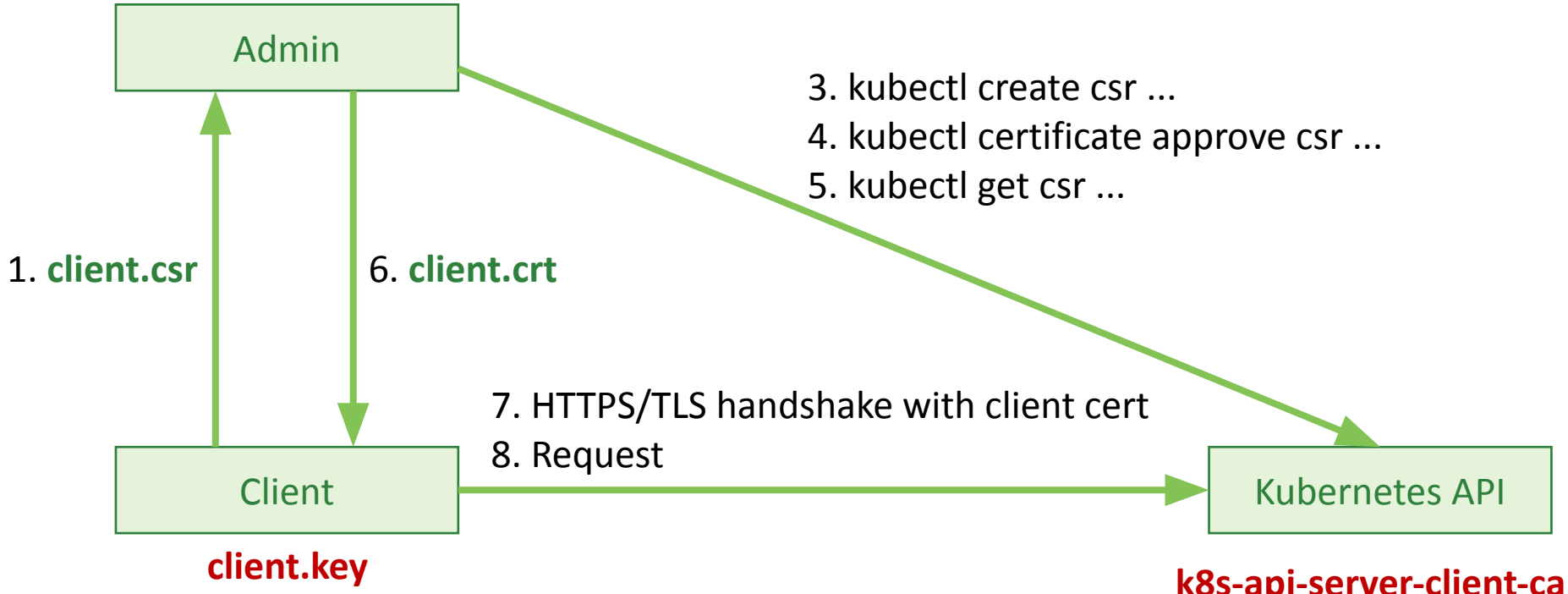
- **Admin:** sign user client cert

```
openssl x509 -req -in user1.csr -CA cluster-ca.crt -CAkey cluster-ca.key \
    -set_serial 101 -extensions client -days 365 -outform PEM -out user1.crt
```

- **User:** use with kubectl via options or kubeconfig

```
kubectl --client-key=user1.key --client-certificate=user1.crt get nodes

kubectl config set-credentials user1 --client-key user1.key --client-certificate user1.crt --embed-certs
kubectl config set-context user1 --cluster demo-rbac --user user1
kubectl --context=user1 get nodes

kubectl config use-context user1
kubectl config get-contexts
kubectl get nodes
```

Authentication: X509 Client Cert, K8S CSR




```
--client-ca-file=k8s-api-server-client-ca.key
```

Authentication: X509 Client Cert, K8S CSR

- **User:** generate user CSR

```
openssl req -new -key user2.key -out user2.csr -subj "/CN=user2/O=group1/O=group2"
```

- **Admin:** use Kubernetes API server to sign the CSR

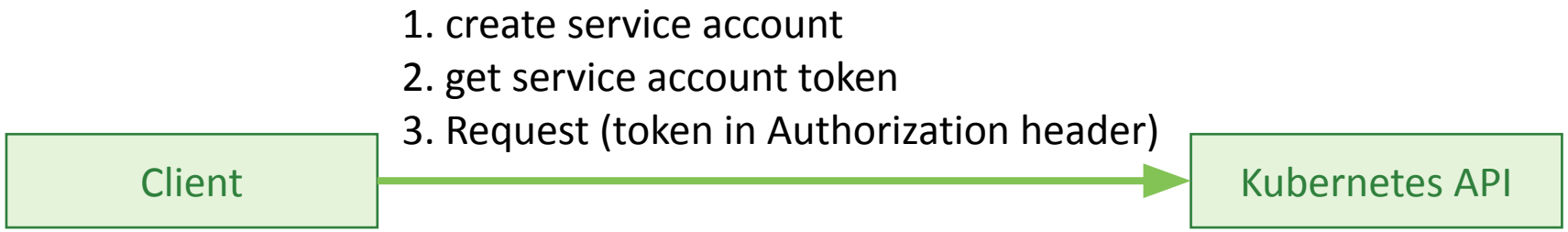
```
kubectl apply -f - <<EOF
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: user2
spec:
  request: $(cat user2.csr | base64 | tr -d '\n')
  usages: ['digital signature', 'key encipherment',
    'client auth']
EOF
```

```
kubectl certificate approve user2
kubectl certificate deny user2

kubectl get csr user2 -o jsonpath='{.status.certificate}' | \
    base64 --decode > user2.crt
```

- **User:** use with kubectl via options or kubeconfig

```
kubectl --client-key=user2.key --client-certificate=user2.crt get nodes

kubectl config set-credentials user2 --client-key user2.key --client-certificate user2.crt --embed-certs
kubectl config set-context user2 --cluster demo-rbac --user user2
```

Authentication: Service Account



Authentication: Service Account, Example

- Create service account

```
kubectl create serviceaccount sa1
```

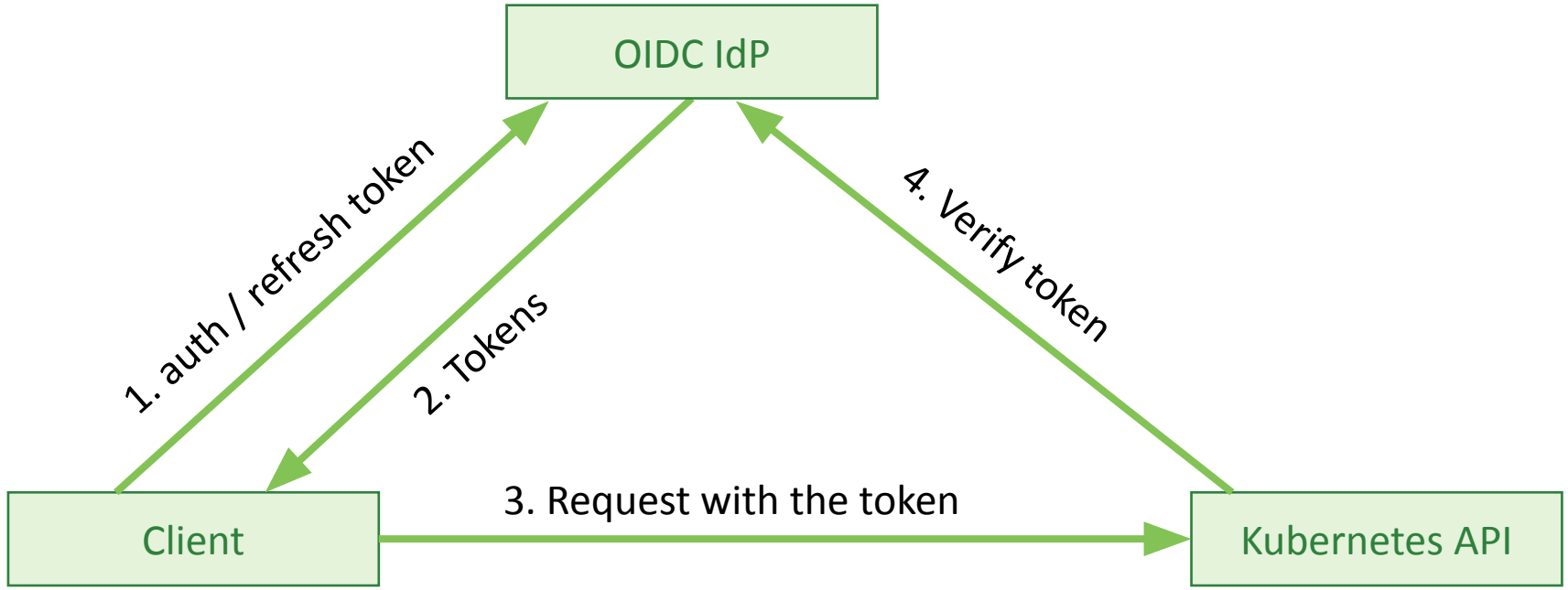
- Get service account token

```
kubectl get –o yaml sa sa1
SA_SECRET="$(kubectl get sa sa1 -o jsonpath='{.secrets[0].name}')"

kubectl get -o yaml secret "${SA_SECRET}"
SA_TOKEN="$(kubectl get secret "${SA_SECRET}" -o jsonpath='{.data.token}' | base64 -d)"
```

- Send request

```
kubectl "--token=${SA_TOKEN}" get nodes

kubectl config set-credentials sa1 "--token=${SA_TOKEN}"
kubectl config set-context sa1 --cluster demo-rbac --user sa1
```

Authentication: OIDC




```
--oidc-client-id=kubernetes
--oidc-groups-claim=user_groups
--oidc-issuer-url=https://idp.example.com
--oidc-username-claim=preferred_username
```

Authentication: OIDC, Demo

Kublr uses Keycloak by default

- Multiple “realms”
- OIDC, SAML, LDAP, AD, Kerberos support
- User federation and Identity Broker support

Demo

- “demo-app” realm
- “kubernetes” OIDC client
- “demo-rbac” Kublr cluster with OIDC auth configured for the client

```
spec:
  master:
    kublrAgentConfig:
      kublr:
        kube_api_server_flag:
          oidc_client_id: '--oidc-client-id=kubernetes'
          oidc_groups_claim: '--oidc-groups-claim=user_groups'
          oidc_issuer_url: '--oidc-issuer-url=https://***'
          oidc_username_claim: '--oidc-username-claim=preferred_username'
```

Authentication: OIDC, Example

Login (visualization)

```
curl \
  -d "grant_type=password" \
  -d "scope=openid" \
  -d "client_id=kubernetes" \
  -d "client_secret=${CLIENT_SECRET}" \
  -d "username=da-admin" \
  -d "password=${USER_PASSWORD}" \
  https://kcp.kublr-demo.com/auth/realms/demo-app/protocol/openid-connect/token | jq .
```

Login (CLI)

```
eval "$(curl -d "grant_type=password" -d "scope=openid" -d "client_id=kubernetes" \
  -d "client_secret=${CLIENT_SECRET}" -d "username=da-admin" -d "password=${USER_PASSWORD}" \
  https://kcp.kublr-demo.com/auth/realms/demo-app/protocol/openid-connect/token | \
  jq -r '"REFRESH_TOKEN="+.refresh_token,"TOKEN="+.access_token,"ID_TOKEN="+.id_token')" ; \
  echo ; echo "TOKEN=${TOKEN}" ; echo ; echo "ID_TOKEN=${ID_TOKEN}" ; echo ; \
  echo "REFRESH_TOKEN=${REFRESH_TOKEN}" ; echo
```

Authentication: OIDC, Example

Refresh (visualization)

```
curl \
  -d "grant_type=refresh_token" \
  -d "client_id=kubernetes" \
  -d "client_secret=${CLIENT_SECRET}" \
  -d "refresh_token=${REFRESH_TOKEN}" \
  https://kcp.kublr-demo.com/auth/realms/demo-app/protocol/openid-connect/token | jq -r .
```

Refresh (CLI)

```
eval "$(curl -d "grant_type=refresh_token" -d "client_id=kubernetes" \
  -d "client_secret=${CLIENT_SECRET}" -d "refresh_token=${REFRESH_TOKEN}" \
  https://kcp.kublr-demo.com/auth/realms/demo-app/protocol/openid-connect/token | \
  jq -r '"REFRESH_TOKEN="+.refresh_token,"TOKEN="+.access_token,"ID_TOKEN="+.id_token')" ; \
  echo ; echo "TOKEN=${TOKEN}" ; echo ; echo "ID_TOKEN=${ID_TOKEN}" ; echo ; \
  echo "REFRESH_TOKEN=${REFRESH_TOKEN}"
```

Authentication: OIDC, Example

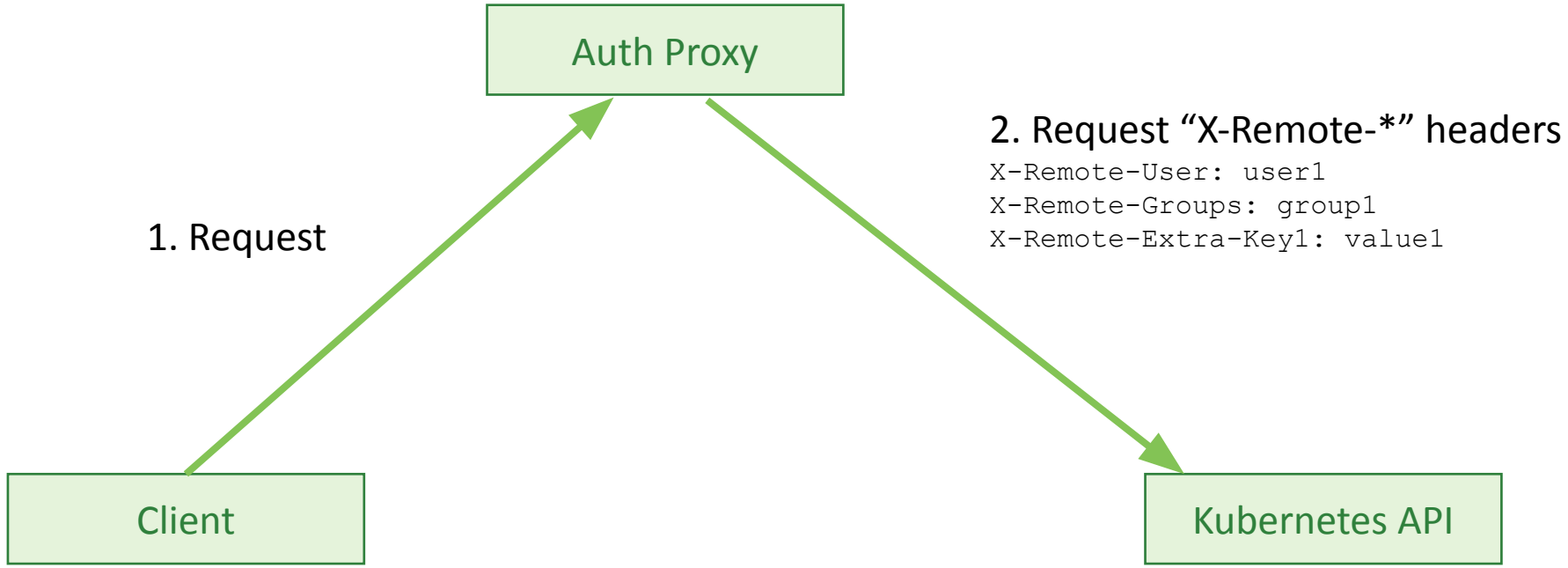
Token introspection

```
curl \
  --user "kubernetes:${CLIENT_SECRET}" \
  -d "token=${TOKEN}" \
  https://kcp.kublr-demo.com/auth/realms/demo-app/protocol/openid-connect/token/introspect | jq .
```

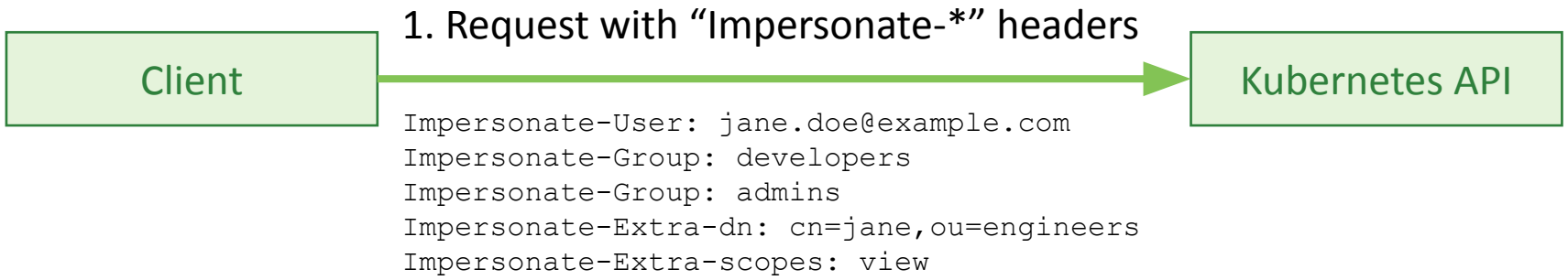
kubectl configuration

```
kubectl config set-credentials da-admin \
   "--auth-provider=oidc" \
   "--auth-provider-arg=idp-issuer-url=https://kcp.kublr-demo.com/auth/realms/demo-app" \
   "--auth-provider-arg=client-id=kubernetes" \
   "--auth-provider-arg=client-secret=${CLIENT_SECRET}" \
   "--auth-provider-arg=refresh-token=${REFRESH_TOKEN}" \
   "--auth-provider-arg=id-token=${ID_TOKEN}"

kubectl config set-context da-admin --cluster=demo-rbac --user=da-admin

kubectl --context=da-admin get nodes
```

Authentication: Authenticating Proxy



Authentication: Impersonation




```
kubectl get nodes \
    --as "system:serviceaccount:default:sa1" \
    --as-group g1 \
    --as-group g2
```

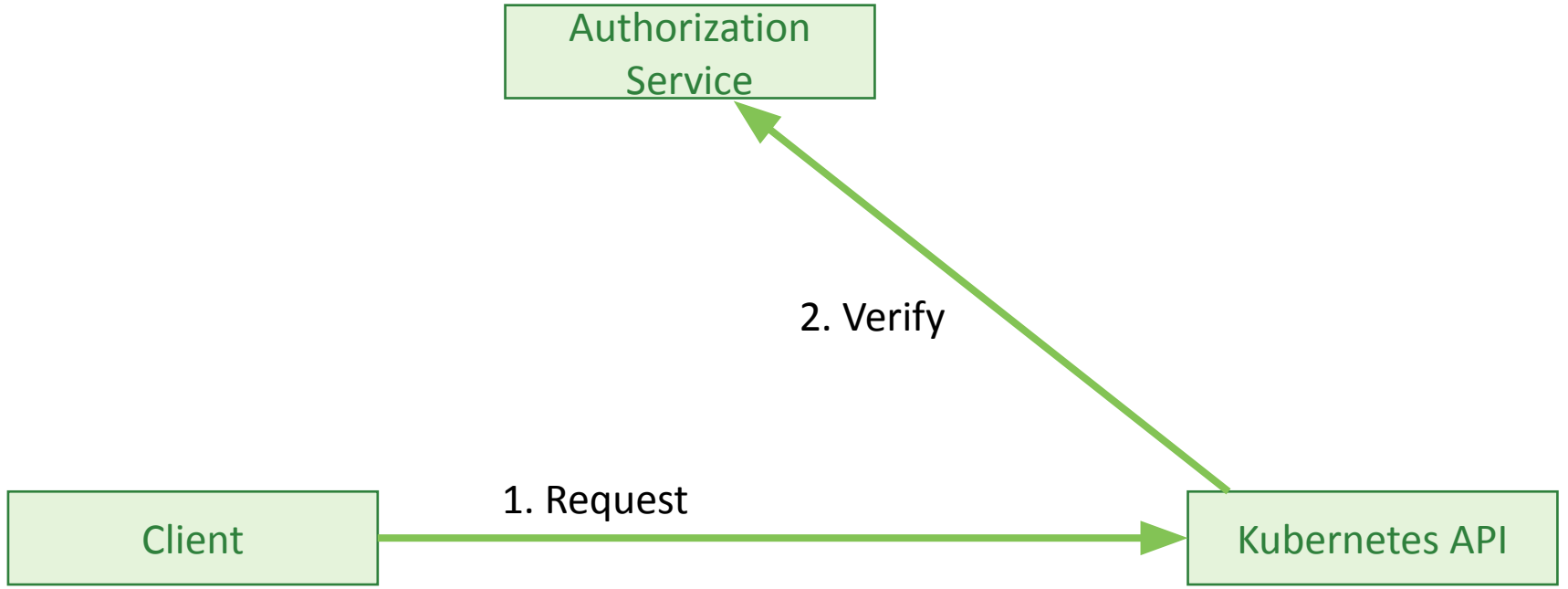
Authorization: Mechanisms

Mechanism	Decision source	Usage
Node	API Server built-in	Internal use (kubelets)
ABAC	Static file	Insecure, deprecated
RBAC	API Objects	Users and administrators
WebHook	External services	Integration
AlwaysDeny AlwaysAllow	API Server built-in	Testing

Authorization, tools

- `kubectl auth can-i ...`
- `kubectl whoami`
- `kubectl --v=8 ...`

Authorization: WebHook




```
--authorization-webhook-config-file=auth-cfg.yaml
```

Config file in kubeconfig format

Authorization: RBAC

Three groups

Subjects

User 

Group 

Service Account 

Operations

list	
get	get
create	post
update	put
delete	delete
watch	
patch	

Resources

Pods
Nodes
ConfigMaps
Secrets
Deployments
…

Connected through RBAC

Roles and ClusterRoles

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role

metadata:
  namespace: default
  name: role1

rules:

- apiGroups: ['*']
  resources: ['nodes', 'pods', 'pods/log']
  verbs: ['get', 'list']

- apiGroups: ['*']
  resources: ['configmaps']
  resourceNames: ['my-configmap']
  verbs: ['get', 'list']
```

- Roles and ClusterRoles define a set of allowed actions on resources
- Role is namespaced
- Cannot include non-namespaces resources or non-resource URLs

ClusterRoles

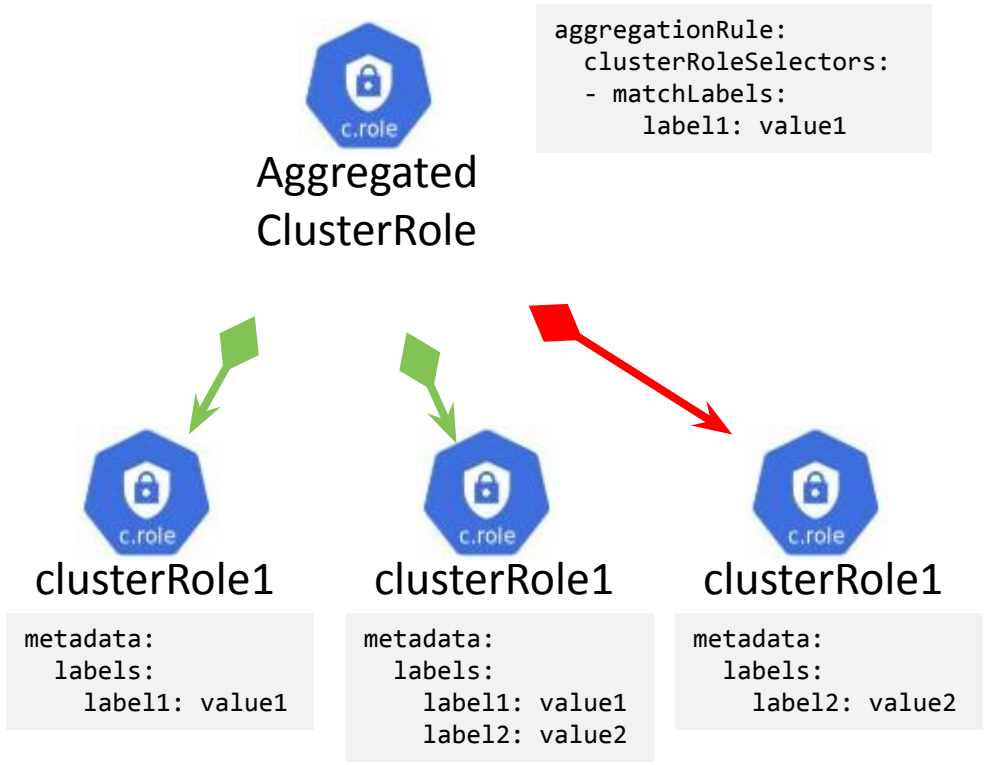
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole

metadata:
  namespace: default
  name: clusterRole1

rules:

- apiGroups: ['*']
  resources: ['nodes', ‘pods']
  verbs: ['get', 'list']

- nonResourceURLs: ['/api', '/healthz*']
  verbs: ['get', 'head']
```

- ClusterRole is not namespaced
- non-namespaced resources access
- non-resource URLs access

Aggregated ClusterRoles

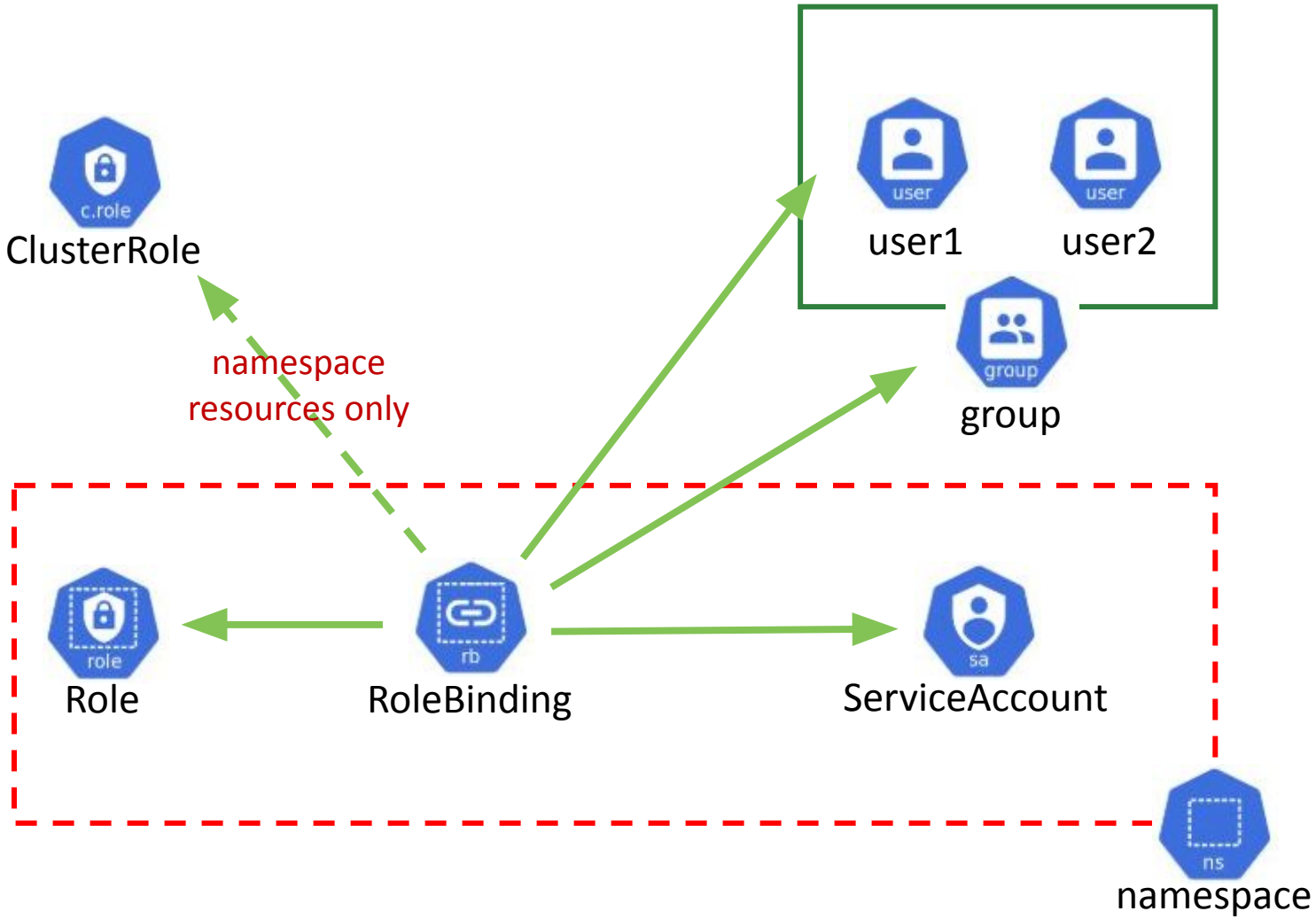
Aggregated ClusterRole combines rules from other cluster roles

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole

metadata:
  name: aggregatedClusterRole1

aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      label1: value1

# The control plane automatically fills in the rules
rules: []
```



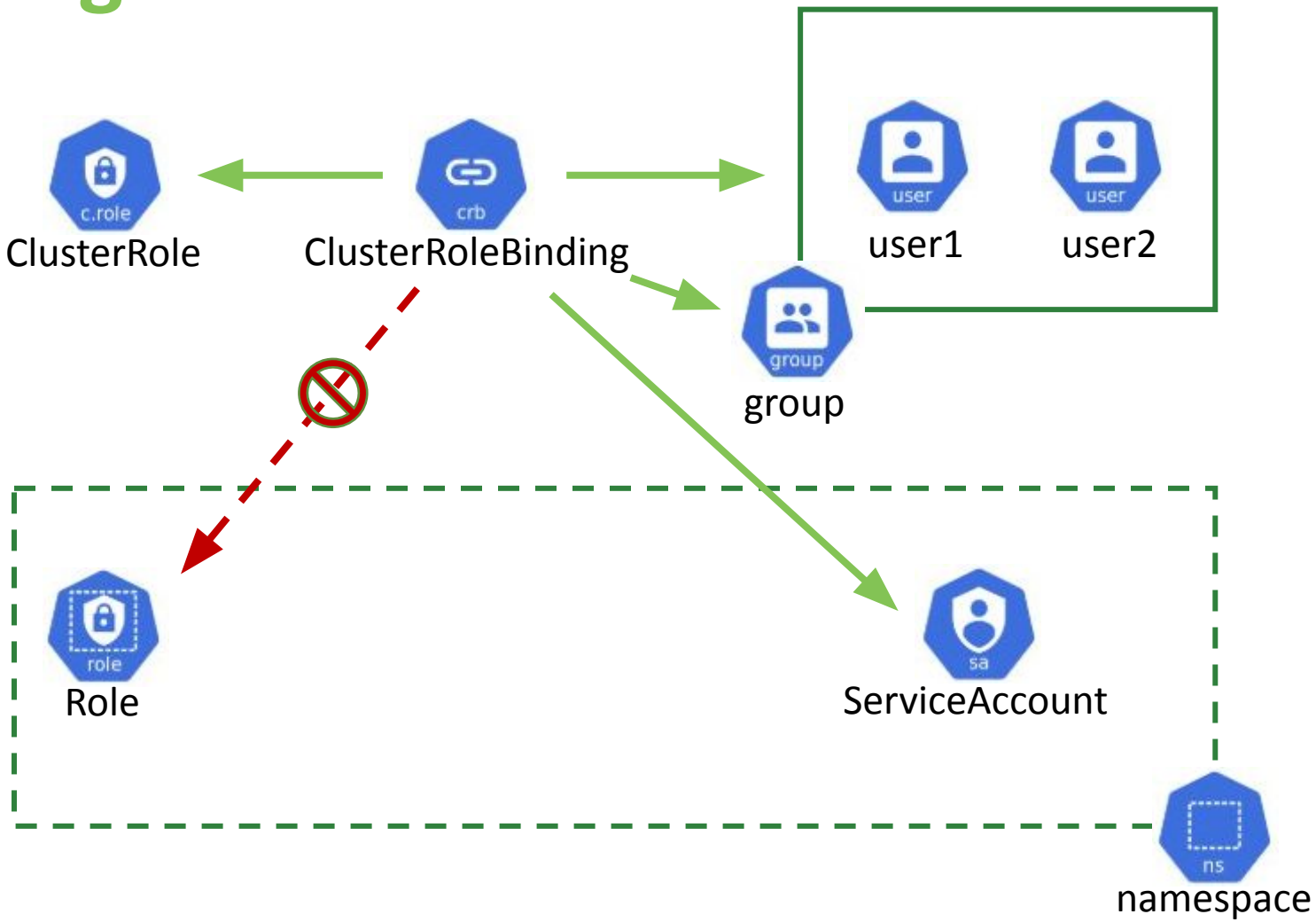
RoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: roleBinding1
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: role1
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: user1
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: group1
- kind: ServiceAccount
  name: sa1
  namespace: default
```



ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: roleBinding1
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: clusterRole1
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: user1
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: group1
- kind: ServiceAccount
  name: sa1
  namespace: default
```



Built-in ClusterRoles

Role	Access	Can do
cluster-admin	Full access to all cluster resources	Anything in the cluster
admin	Full access to all namespace resources	Anything in a namespace
edit	Full access to all namespace resources except `Role`, `RoleBinding`, `LocalSubjectAccessReviews`	Anything in a namespace except granting and checking access
view	RO access to all namespace resources except sensitive ones	View and list non-sensitive objects in a namespace

Use-cases

- Start with build-in roles
  - Cluster admin
  - Namespace admin
  - Namespace developer
  - Namespace read-only user
- Define new roles as needed
- Beware of “gotchas”

“Gotchas”

- Privilege escalation via pod creation
- Non-namespaced objects
  - CRD, PriorityClass, PodSecurityPolicy
  - Often needed for development, especially in advanced DevOps / SRE culture
  - As a result, developers need self-service dev cluster management capabilities
- Role and role binding conventions
  - Name
  - Role bindings – one per subject, one per role, or mixed

Next steps

- PodSecurityPolicy
- NetworkPolicy
- Limits and Quotas
- Admission Controllers
- Dynamic admission control
- Dynamic policies, OPA

Kubernetes API Groups and Objects

API Group	API Obects
rbac.authorization.k8s.io/v1	ClusterRole Role ClusterRoleBinding RoleBinding
authentication.k8s.io/v1	TokenReview
admissionregistration.k8s.io/v1	MutatingWebhookConfiguration ValidatingWebhookConfiguration
authorization.k8s.io/v1	LocalSubjectAccessReview SelfSubjectAccessReview SelfSubjectRulesReview SubjectAccessReview
certificates.k8s.io/v1beta1	CertificateSigningRequest
policy/v1beta1	PodSecurityPolicy

References

https://github.com/rajatjindal/kubectl-whoami

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

https://kubernetes.io/docs/concepts/cluster-administration/certificates/

https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/

Q&A

@olgch; @kublr

Weekly Virtual Event

Virtual Meetups

- Small intimate meetups
- With attendee intro round (ideally on camera)
- Open discussion at the end

Tomorrow 4pm ET: **A No-BS Checklist for Enterprise-grade Kubernetes Deployments**

Meetup.com group: **All Things Kubernetes Meetup & Happy Hour**

Webinars

- Traditional webinar

Tomorrow: **Designing reliable, self-healing cloud native apps**

Next week Thu: **Cloud Abstraction, the Often Overlooked Power of Kubernetes**

Register on our website (under resources)

kublr

Signup for our newsletter at kublr.com

Oleg Chunikhin

CTO

oleg@kublr.com

@olgch

Kublr | kublr.com

@kublr

@olgch; @kublr