Kubernetes - Part 4

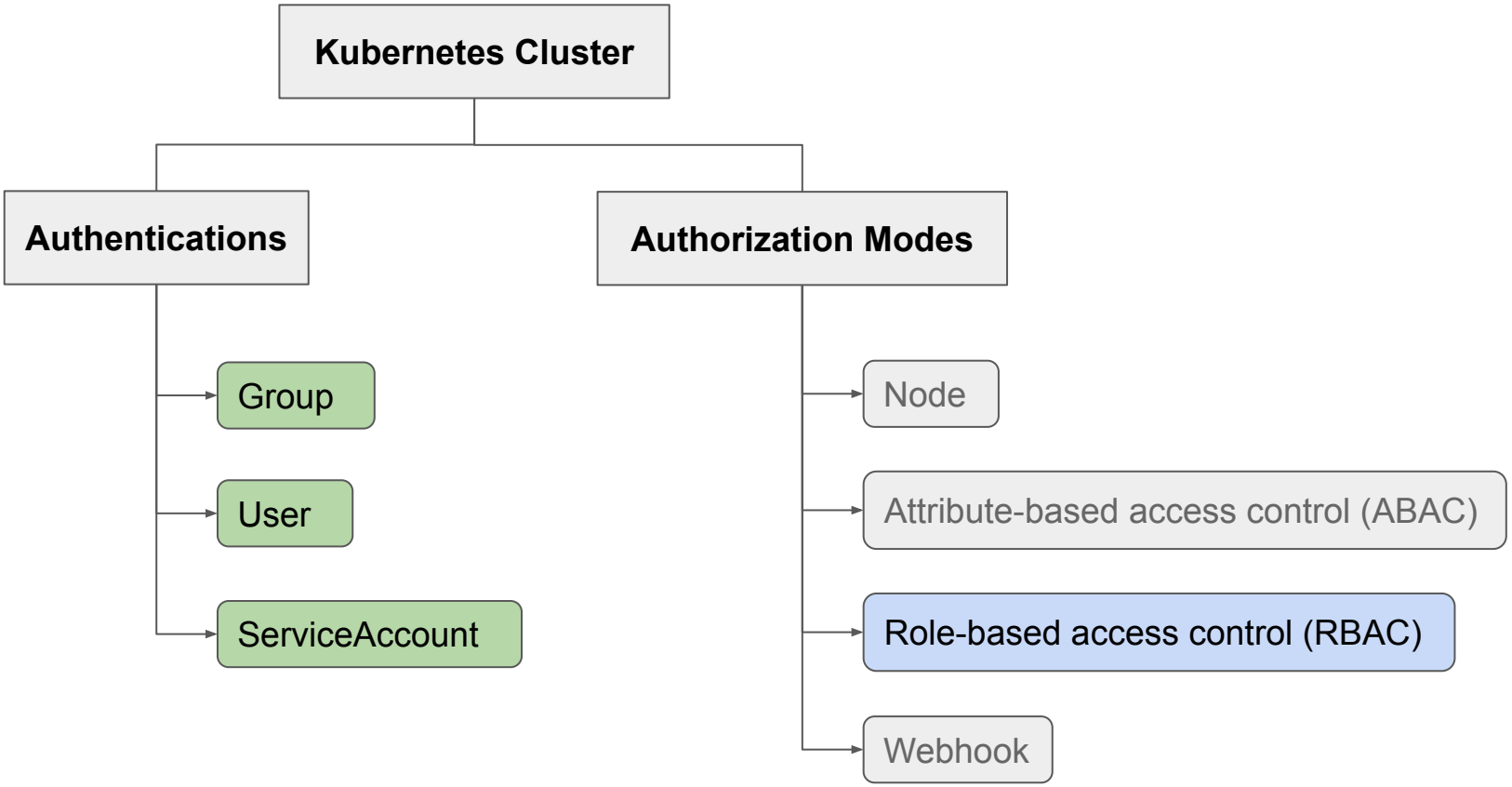
Solar Team

Version: 1.0

Agenda

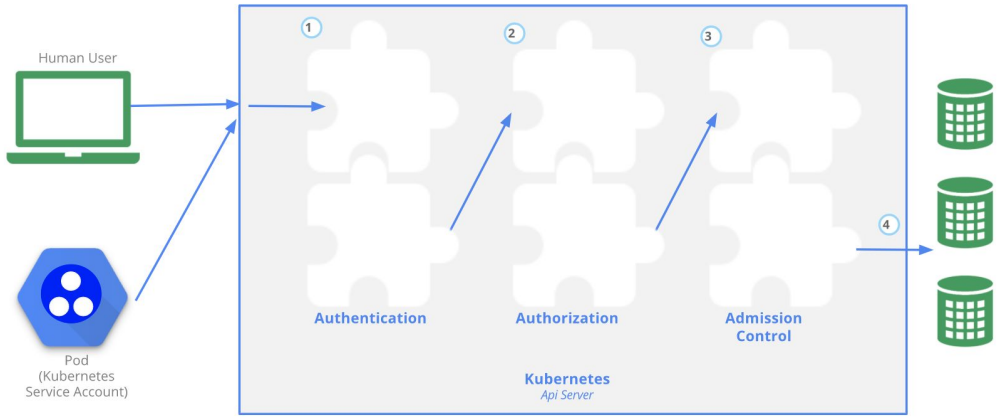
- Kubernetes Authentication
  - User and Group
  - Service Account
- Kubernetes Authorization
  - RBAC Authorization
    - Role
    - Role Binding
    - ClusterRole
    - ClusterRoleBinding

Kubernetes API Access Control



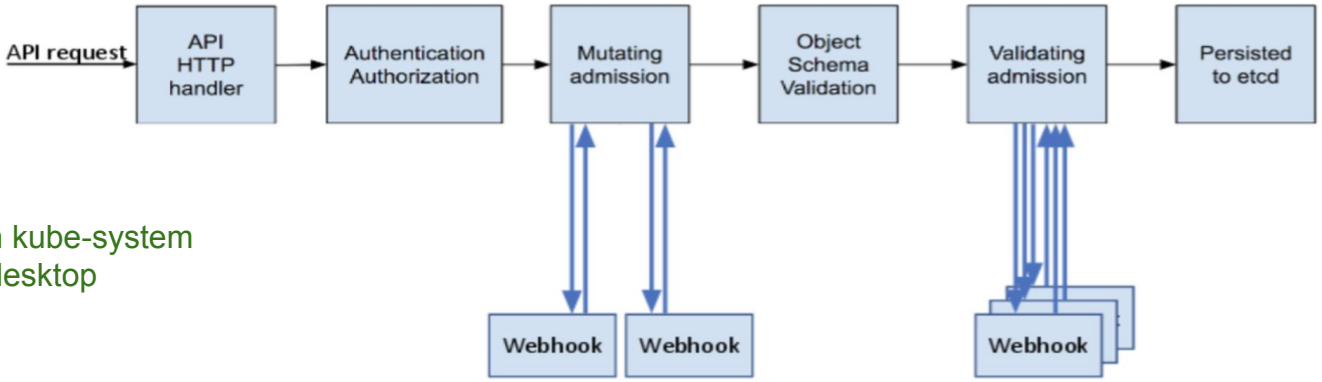
Kubernetes Authentication

- **All actions in a Kubernetes cluster need to be authenticated and authorized.**
- Kubernetes clusters have two categories of users:
  - User accounts - for humans (Kubernetes does not have objects which represent normal user accounts.)
  - Service accounts - for processes (which run in pods), managed by Kubernetes.
- Support many authentication strategies likes, uses client certificates, bearer tokens, an authenticating proxy, or HTTP basic auth.
- Support anonymous users.



Admission Controllers

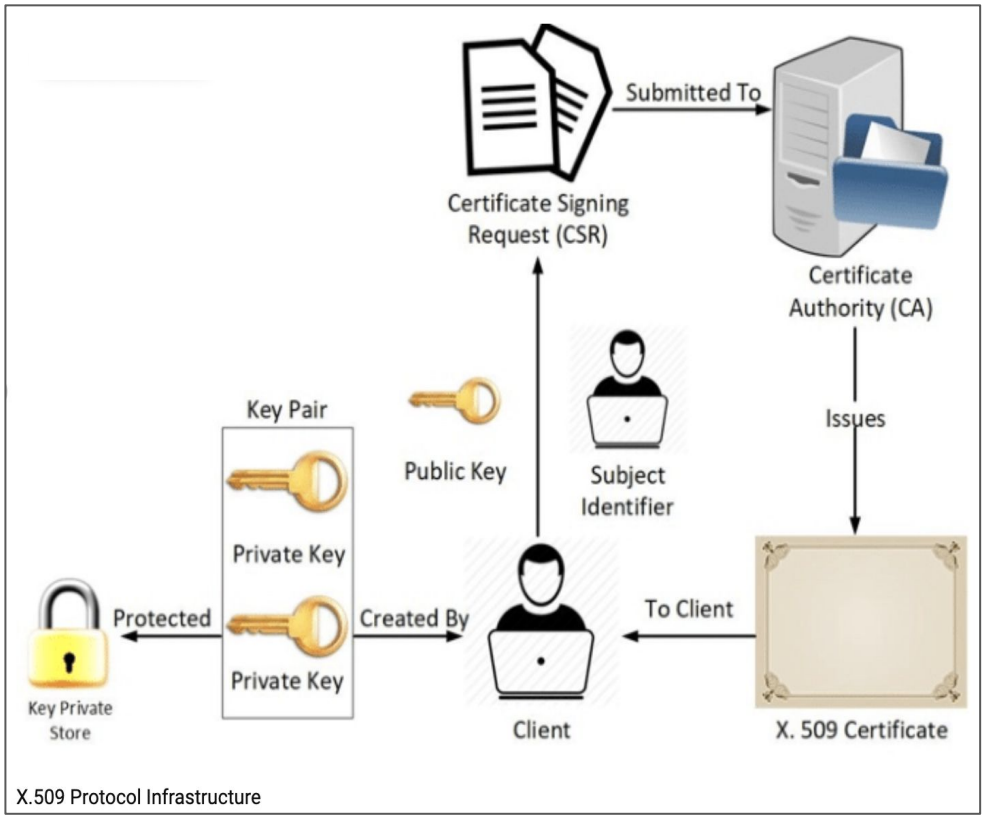
- An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized.
- Kubernetes admission controllers are plugins that govern and enforce how the cluster is used.
- More than 30 admission controllers shipped with Kubernetes.
- Example: ServiceAccount, AlwaysPullImages, MutatingAdmissionWebhook and ValidatingAdmissionWebhook.



Admission Controller Phases

- kubectl describe pods -n kube-system kube-apiserver-docker-desktop

Create New Kubernetes User - X.509 Client Certificates



X.509 Protocol Infrastructure

Source: https://www.researchgate.net/figure/X509-Protocol-Infrastructure_fig3_320093430

- The Kubernetes Certificates API provide a mechanism to obtain x509 certificates by submitting a certificate signing request, and having it asynchronously approved and issued.
- This API can be used to request client certificates to authenticate to kube-apiserver (with the "kubernetes.io/kube-apiserver-client" signerName)
- FEATURE STATE: Kubernetes v1.19 [stable]

Creat New Kubernetes User - Prepare CSR

- To create users with X.509 client certificates and how to manage authorizations with the basic Kubernetes Role-based access control (RBAC) API Objects.
- Create the client key
  - openssl genrsa -out anurak.key 2048
- Create a certificate signing request(CSR)
  - openssl req -new -key **anurak.key** -out anurak.csr
  - Or openssl req -new -key anurak.key -out user1.csr -subj "/CN=anurak/O=dev”
  - CN is the name of the user and O is the group that this user will belong to.

Creat New Kubernetes User - Submit CSR

```yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
 name: anurak
spec:
 groups:
 - system:authenticated
 signerName: kubernetes.io/kube-apiserver-client
 request: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBS...Cg==
 usages:
 - client auth
```

csr-anurak.yml

- Create a CertificateSigningRequest and submit it to Kubernetes cluster
- “request” contains an x509 certificate signing request encoded in a "CERTIFICATE REQUEST" PEM block. When serialized as JSON or YAML, the data is additionally base64-encoded.
    - cat anurak.csr | base64 | tr -d "\n"
- "kubernetes.io/kube-apiserver-client": issues client certificates that can be used to authenticate to kube-apiserver.
- kubectl apply -f csr-anurak.yml

Creat New Kubernetes User - Approve Or Deny CSR

- Approve the CSR:
  - kubectl certificate approve anurak
- Or deny the CSR
  - kubectl certificate deny anurak
- Retrieve the certificate from the CSR:
  - kubectl get csr/myuser -o yaml
- Export the issued certificate from the CertificateSigningRequest. The certificate value is in Base64-encoded format under status.certificate.
  - kubectl get csr myuser -o jsonpath='{.status.certificate}'| base64 -d > anurak.crt

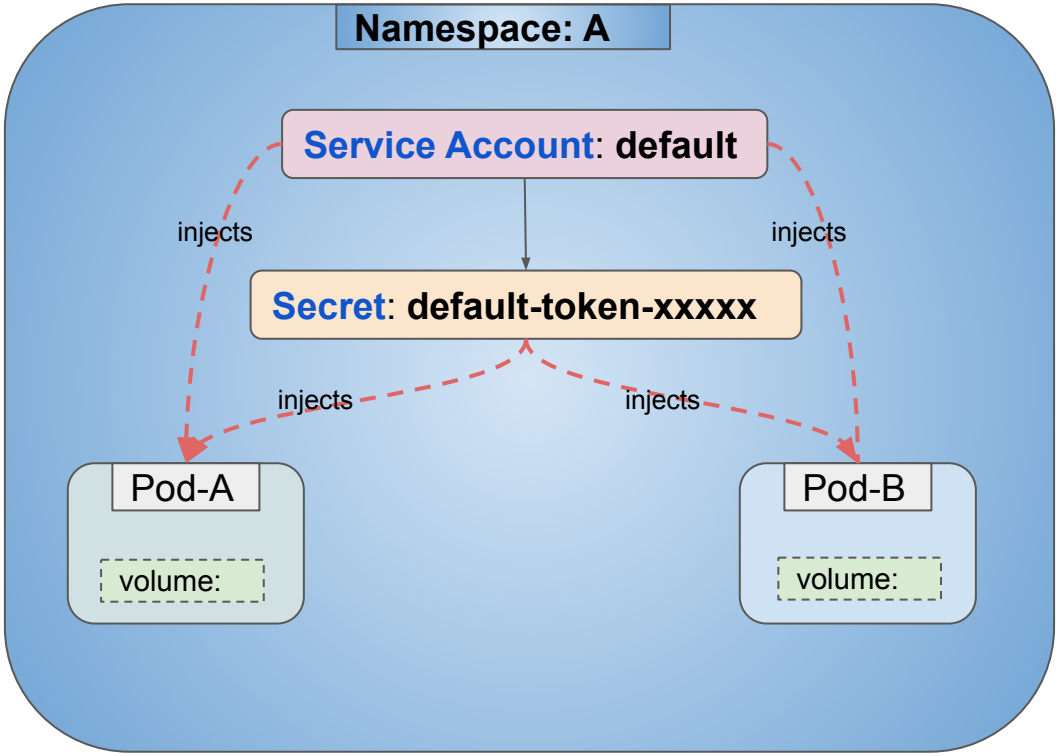
Create New Kubernetes User - Using A New User

- Add this user into the kubeconfig file.
  - kubectl config set-credentials anurak --client-key=anurak.key --client-certificate=anurak.crt --embed-certs=true
- Add the context:
  - kubectl config set-context anurak --cluster=kubernetes --user=anurak
- To test it, change the context to “anurak”:
  - kubectl config use-context anurak
- Create a Role for this new user
  - kubectl create role pod-reviewer --verb=get --verb=list --verb=watch --resource=pods
- Create a RoleBinding for this new user
  - kubectl create rolebinding pod-reviewer --role=pod-reviewer --user=anurak
- Testing
  - kubectl auth can-i get pods
  - kubectl get all
  - kubectl get pods
  - kubectl config use-context docker-desktop # switch user to docker-desktop
  - kubectl get po --user=anurak
  - kubectl get rolebinding -owide | grep anurak

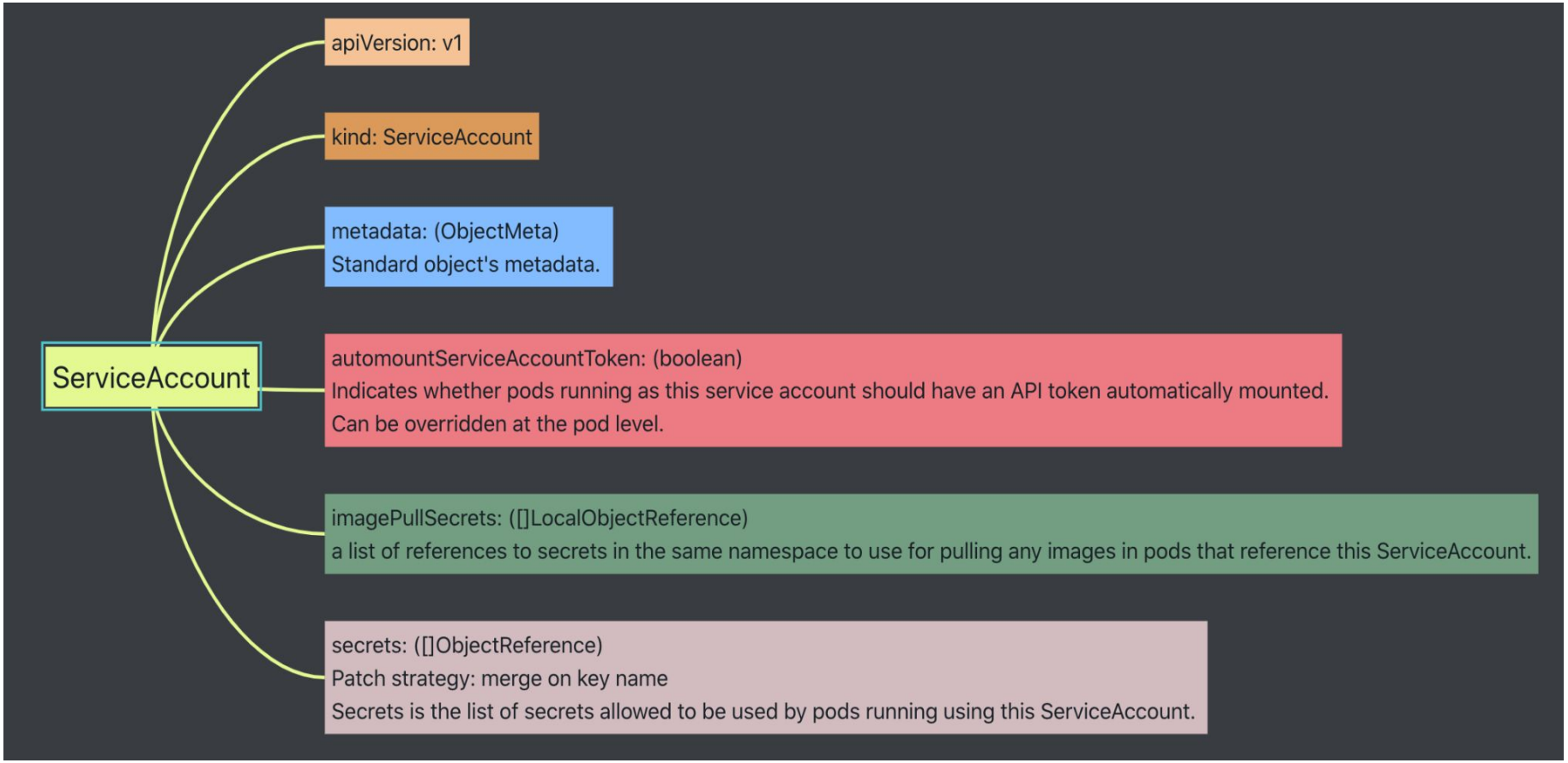
Service Account Overview

- A service account provides an identity for processes that run in a Pod.
- Kubernetes resources that bound to specific namespaces.
- Created automatically by the API server or manually through API calls.
- They are tied to a set of credentials (API credentials) stored as Secrets.
- Every namespace has a default service account resource called “default”.
- A default service account (if we do not specify a service account) is mounted into pods allowing in-cluster processes to talk to the Kubernetes API.
- Listing service accounts
  - kubectl get sa
  - kubectl get secret
  - kubectl get sa default -o yaml
  - kubectl describe secret <default-ac-secret>

Default Service Account

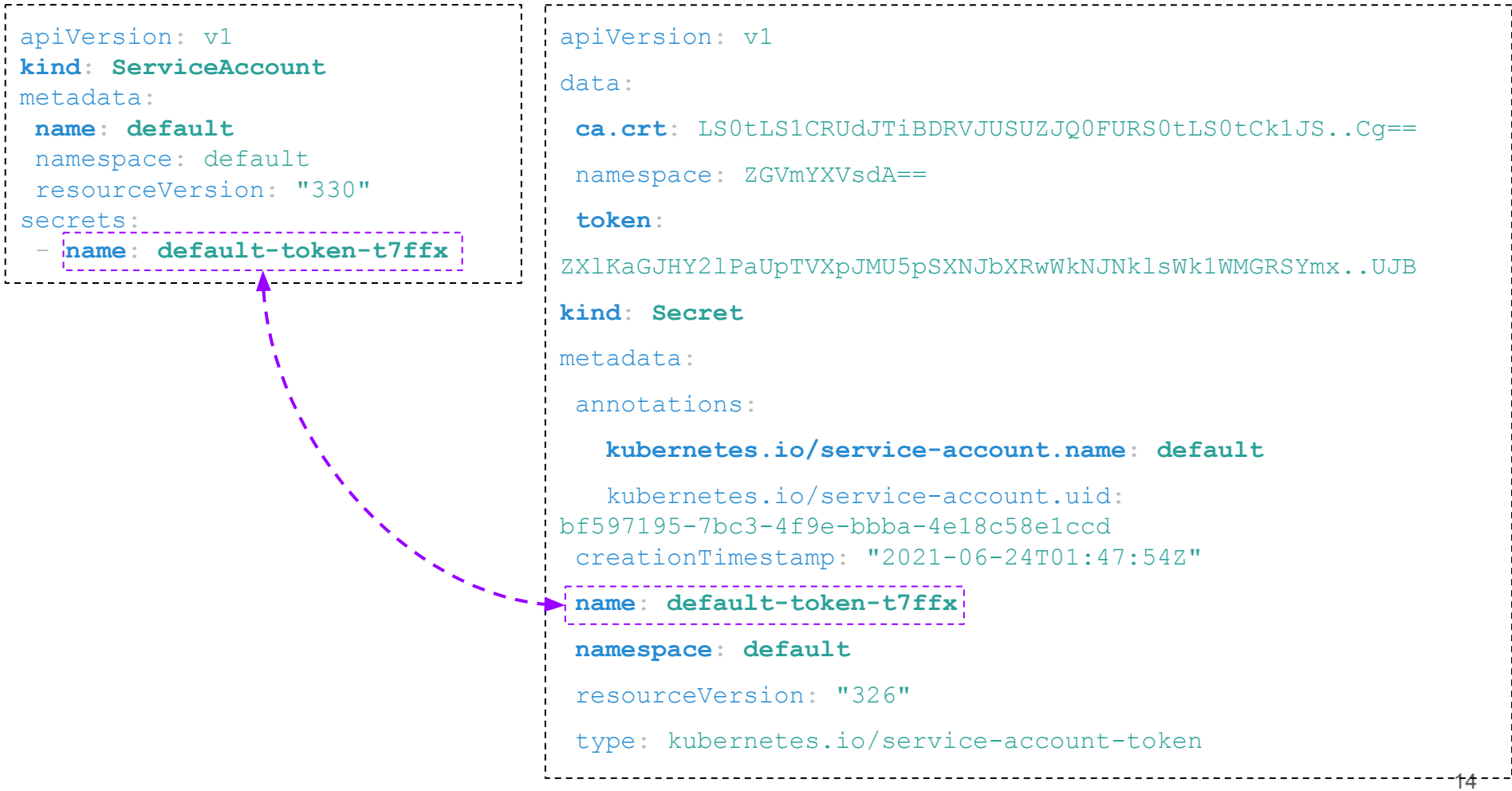


ServiceAccount Object



- kubectl create secret docker-registry -h

Default Service Account and Secret



Create Service Account

- Create a new service account named my-service-account.
  - kubectl create serviceaccount anuraksa
  - kubectl get sa my-service-account -oyaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
 creationTimestamp: "2021-05-05T08:46:57Z"
 name: my-service-account
 namespace: default
 resourceVersion: "2613046"
 selfLink: /api/v1/namespaces/default/serviceaccounts/my-service-account
 uid: 72f7d01d-ad7e-11eb-af4b-025000000001
secrets:
- name: my-service-account-token-hs8gz
```

Automounting ServiceAccount Token

- Not every pod needs to contact the API server.
- From version 1.6+ it is possible to prevent automounting of service account tokens on pods using automountServiceAccountToken: false.
- automountServiceAccountToken can be overridden at the Pod level.

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: nginx-no-token
 namespace: default
spec:
 containers:
 - image: nginx:1.7.9
   imagePullPolicy: IfNotPresent
   name: nginx
   ports:
   - containerPort: 80
     protocol: TCP
   resources: {}
 serviceAccountName: default
 automountServiceAccountToken: false
```

nginx-pod-no-token.yml

- kubectl describe po nginx-no-token
- Checking token file within Pod
  - kubectl exec -it nginx-no-token --sh

“cluster-admin” and “docker-for-desktop-binding”

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 annotations:
   rbac.authorization.kubernetes.io/autoupdate: "true"
 creationTimestamp: "2020-12-15T07:59:03Z"
 labels:
   kubernetes.io/bootstrapping: rbac-defaults
 name: cluster-admin
 resourceVersion: "43"
 selfLink: /apis/rbac.authorization.k8s.io/v1/clusterroles/cluster-admin
 uid: 65cd40ff-3eab-11eb-95d0-025000000001
rules:
- apiGroups:
 - '*'
 resources:
 - '*'
 verbs:
 - '*'
- nonResourceURLs:
 - '*'
 verbs:
 - '*'
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: docker-for-desktop-binding
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: Group
 name: system:serviceaccounts
 namespace: kube-system
```

Using ServiceAccount Examples

- kubectl create serviceaccount anuraksa
- kubectl run -i --tty --rm=true busybox --image=radial/busyboxplus --serviceaccount=anuraksa --restart=Never -- sh
  - TOKEN=$(cat /run/secrets/kubernetes.io/serviceaccount/token)
  - curl -H "Authorization: Bearer $TOKEN" https://kubernetes/api/v1/namespaces/default/pods --insecure
- kubectl delete clusterrolebinding docker-for-desktop-binding
- Test curl again.
- kubectl create role pod-reviewer --verb=get --verb=list --verb=watch --resource=pods
- kubectl create rolebinding pod-reviewer --role=pod-reviewer --serviceaccount anuraksa
- Try again
  - curl -H "Authorization: Bearer $TOKEN" https://kubernetes/api/v1/namespaces/default/pods --insecure
- Bind pop-reviewer role binding to user "anurak"
  - kubectl edit rolebinding pod-reviewer
  - Add these texts
    - - apiGroup: rbac.authorization.k8s.io
    - kind: User
    - name: anurak

Kubernetes Authorization

- Determine whether a request is allowed or denied.
- Multiple authorization modules can be configured and are checked in sequence.
- Authorization Modes.
  - Node - A special-purpose authorization mode that grants permissions to kubelets based on the pods they are scheduled to run.
  - ABAC - Attribute-based access control (ABAC) defines an access control paradigm.
  - RBAC - Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise.
  - Webhook - A WebHook is an HTTP callback: an HTTP POST that occurs when something happens.
- Checking current cluster authorization modes:
  - kubectl cluster-info dump | grep authorization-mode
  - kubectl api-versions
- Checking Kubernetes API access.
  - kubectl auth can-i create deployments --namespace kube-system
  - kubectl auth can-i create deployments --namespace kube-system --as anurak

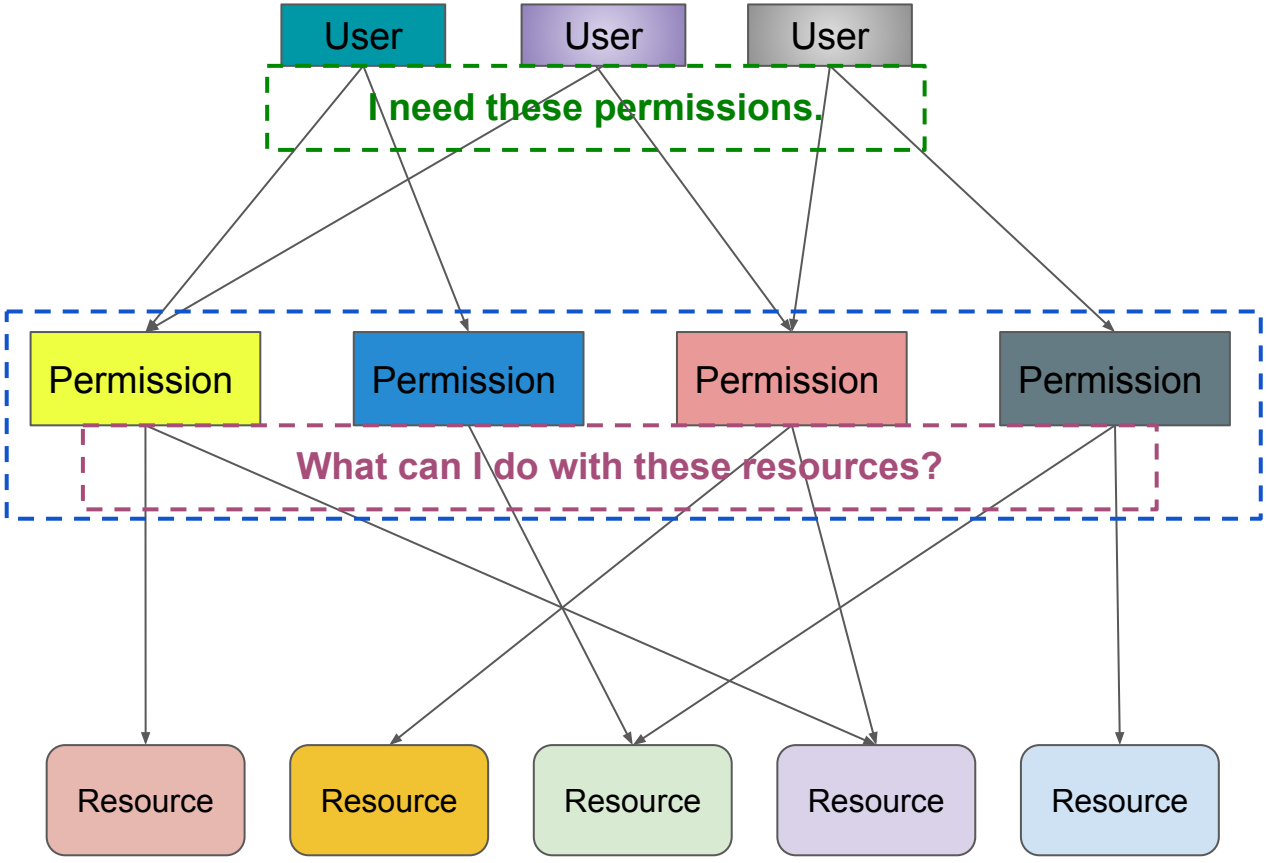
Authorization Modes

- Authorization needs to set up for a new service account.

Authorization Mode	Source	Usage
Node	API Server built-in	kubelet (Internal uses)
ABAC (Attribute-based access control)	local file	Insecure, deprecated
RBAC **(Role-based access control)**	API Objects	Users and administrators
Webhook	External services	Integration
AlwaysDeny AlwaysAllow	API Server built-in	Testing

- Only RBAC and webhook are production-grade authorization.

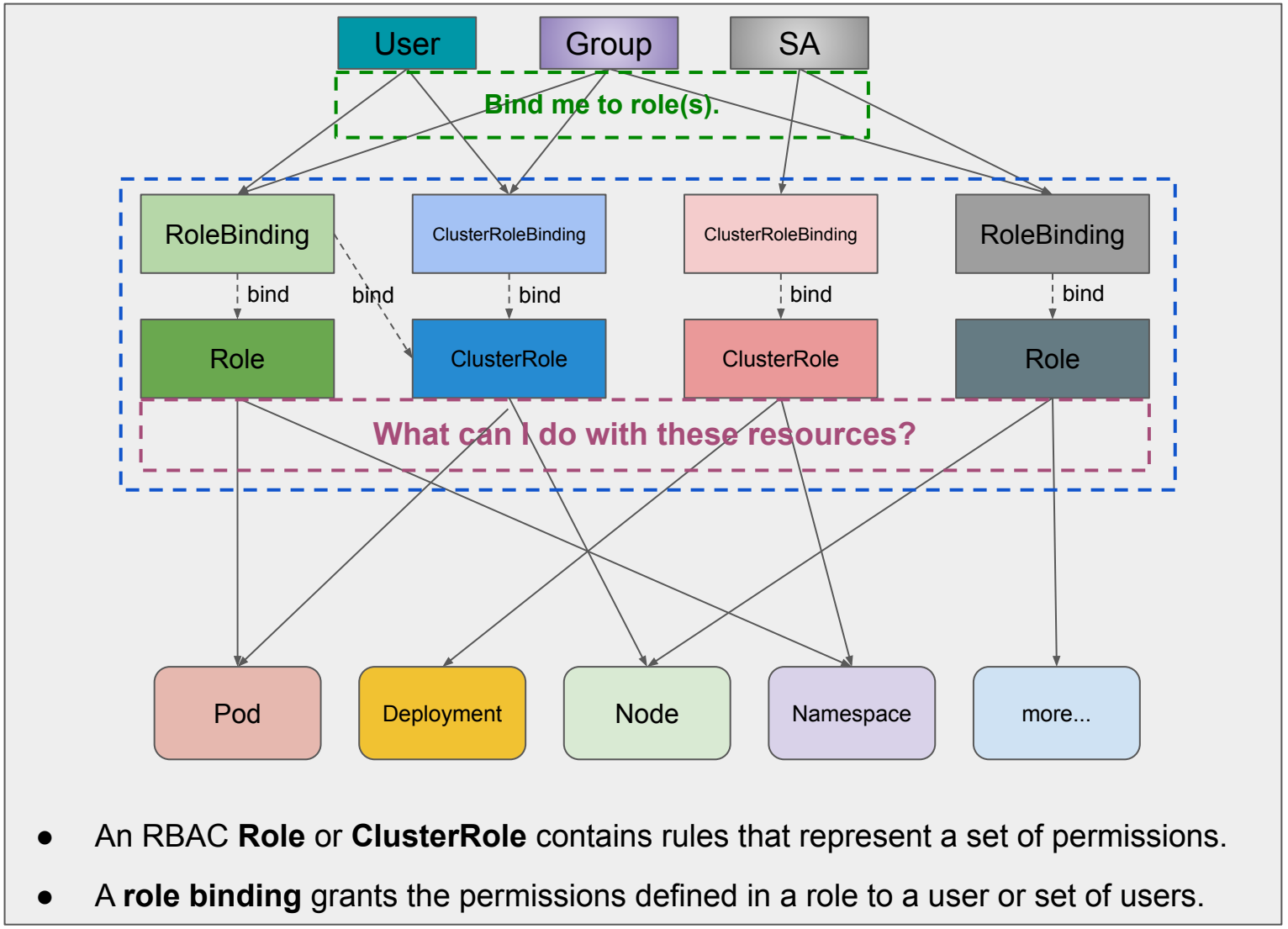
Permission Base Security



RBAC Authorization

- Kubernetes RBAC is a way to define **which users can do what within a Kubernetes cluster**.
- To enable RBAC, start the API server with the --authorization-mode flag
  - kube-apiserver --authorization-mode=RBAC
  - Check current config: kubectl cluster-info dump | grep authorization-mode
- The RBAC API declares **four kinds** of Kubernetes object: Role, ClusterRole, RoleBinding and ClusterRoleBinding.
- **Role or ClusterRole** contains rules that represent a set of permissions.
- **RoleBinding or ClusterRoleBinding** grants the permissions defined in a role to a user or set of users.
- If you want to define a role within a **namespace**, use a Role; if you want to define a role cluster-wide, use a ClusterRole.

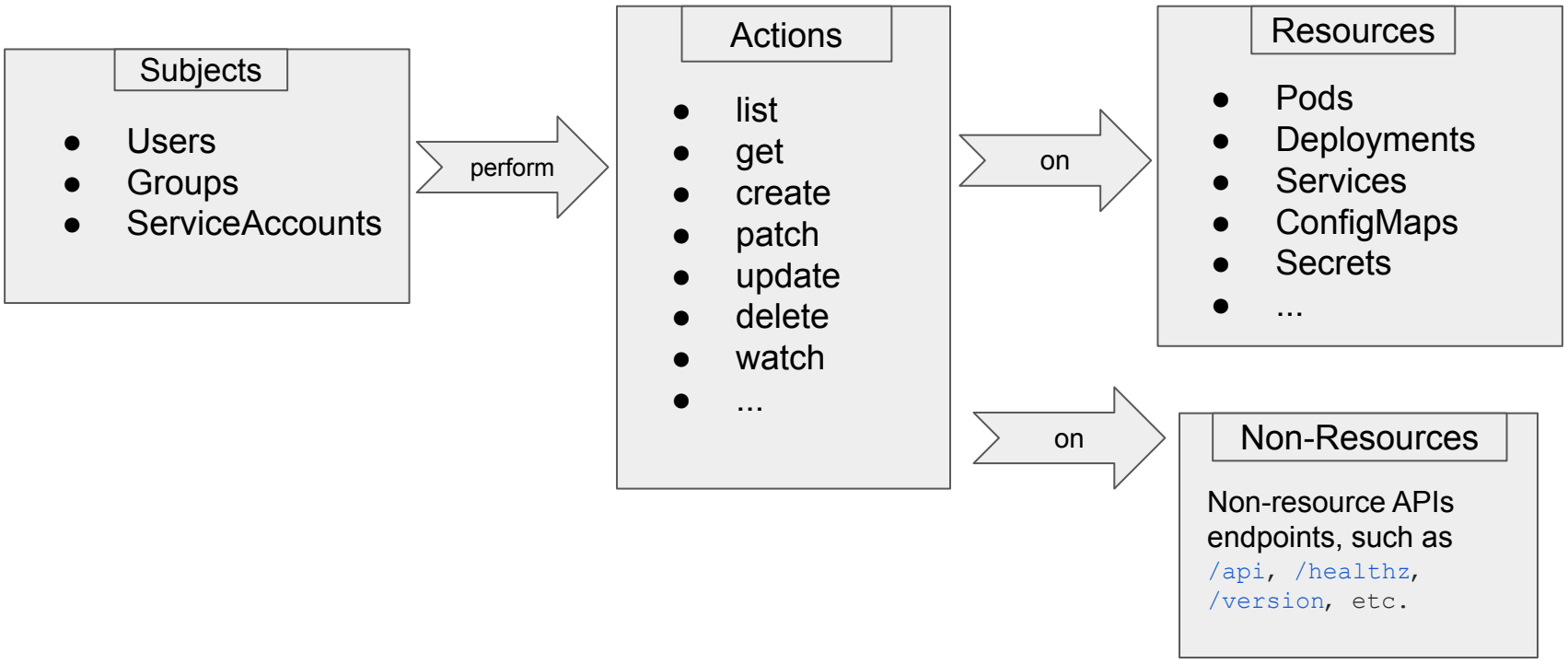
Kubernetes RBAC Authorization



- An RBAC **Role** or **ClusterRole** contains rules that represent a set of permissions.
- A **role binding** grants the permissions defined in a role to a user or set of users.

RBAC Concept - Subjects, Actions and Resources

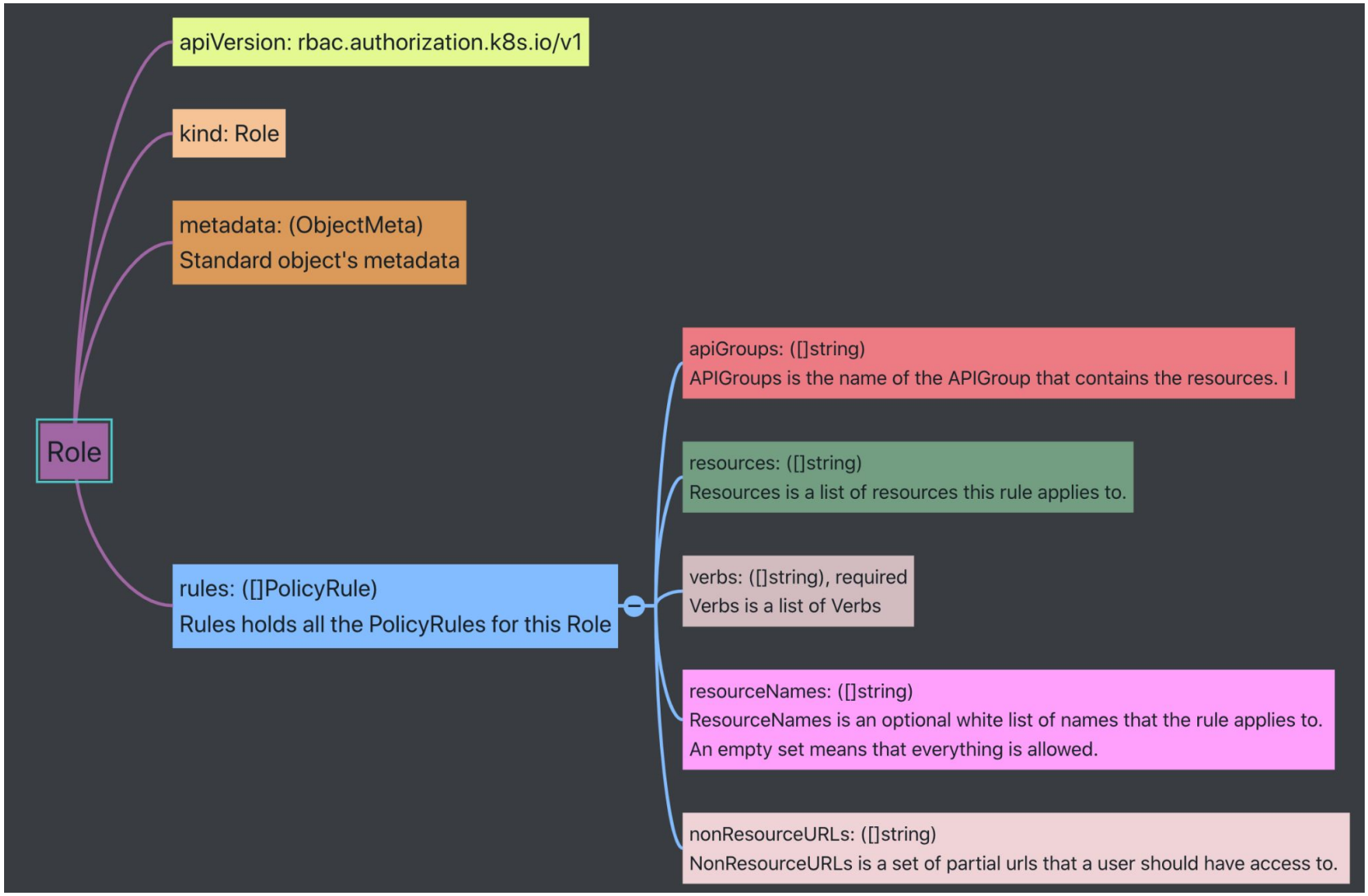
- RBAC allows subjects to perform certain actions on certain resource types.
- Resource actions
  - kubectl api-resources -owide



Role

- **Role** - define the fine-grained permissions that specify a set of resources and actions (verbs) that roles can access and manipulate.
- The permissions can only be granted for the resources that are in the same namespace as the Roles.
- Permissions are purely additive (there are no "deny" rules)
- Rules in Role define a set of permissions for certain operations and resources by specifying apiGroups, resources, and verbs.

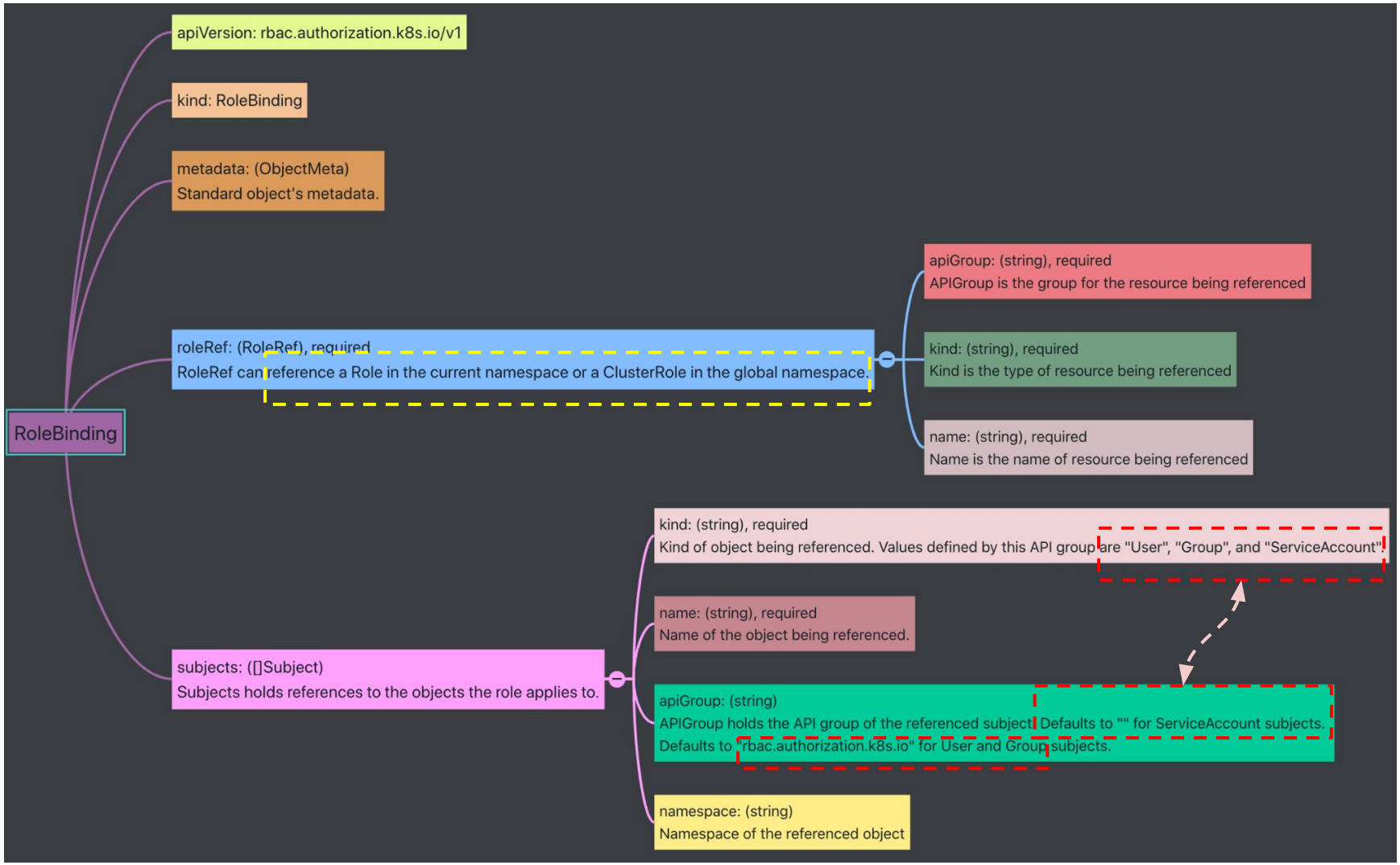
Role Object


apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata: (ObjectMeta)
Standard object's metadata
Role
rules: ([]PolicyRule)
Rules holds all the PolicyRules for this Role
apiGroups: ([]string)
APIGroups is the name of the APIGroup that contains the resources. I
resources: ([]string)
Resources is a list of resources this rule applies to.
verbs: ([]string), required
Verbs is a list of Verbs
resourceNames: ([]string)
ResourceNames is an optional white list of names that the rule applies to.
An empty set means that everything is allowed.
nonResourceURLs: ([]string)
NonResourceURLs is a set of partial urls that a user should have access to.

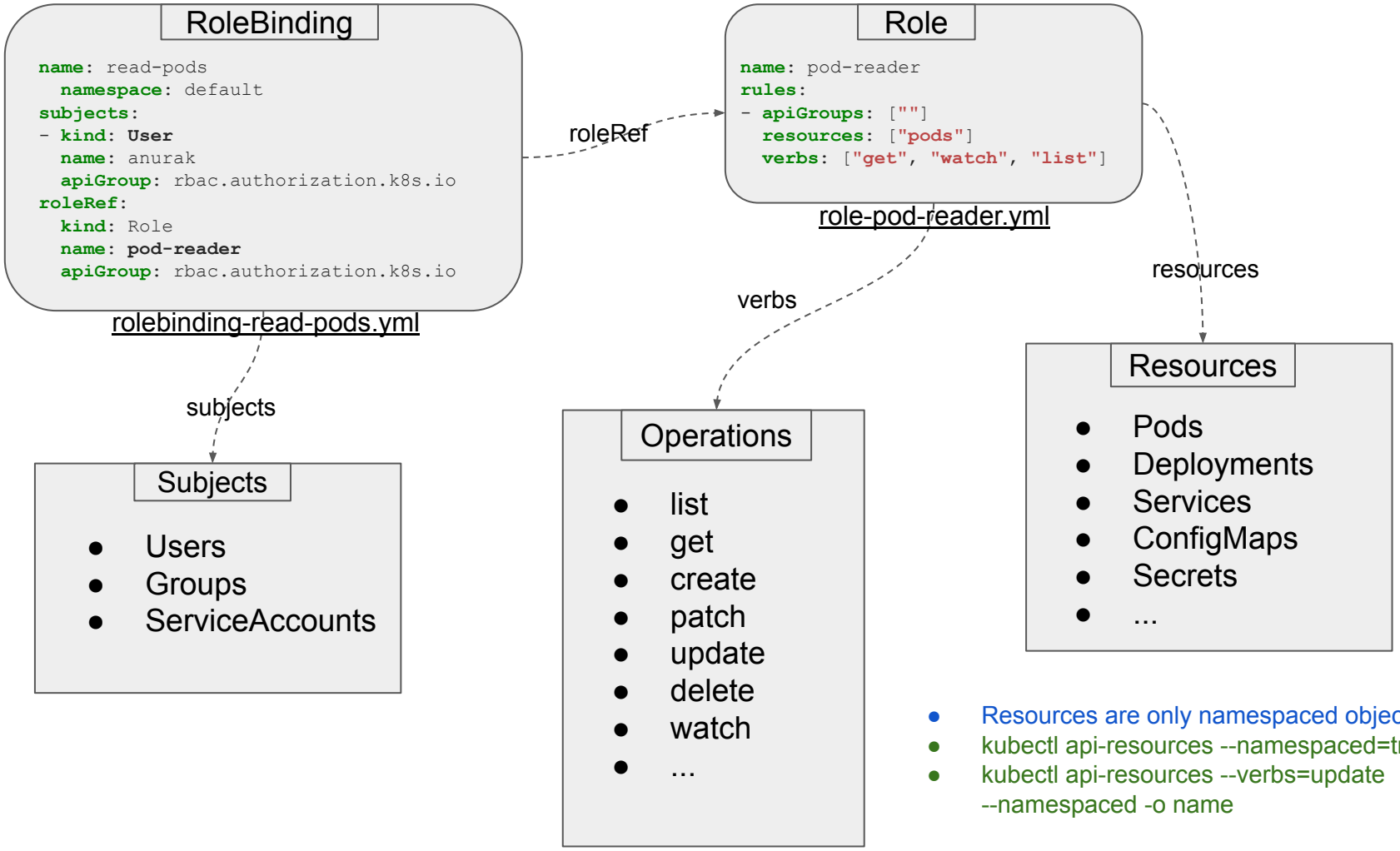
RoleBinding

- **RoleBinding** - grants the permissions defined in a role to a user or set of users.
- A RoleBinding holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted.
- A RoleBinding may reference any Role in the same namespace.
- Alternatively, a RoleBinding can reference a ClusterRole and bind that ClusterRole to the namespace of the RoleBinding

RoleBinding Object



RBAC Roles and RoleBinding



- Resources are only namespaced objects.
- kubectl api-resources --namespaced=true
- kubectl api-resources --verbs=update --namespaced -o name

Note: Check the resource and verb with this command.: kubectl api-resources -owide

Create Role Using kubectl Examples

- Create a Role named "pod-reader" that allows users to perform get, watch and list on pods:
  - kubectl create role pod-reader --verb=get --verb=list --verb=watch --resource=pods
- Create a Role named "pod-reader" with resourceNames specified:
  - kubectl create role pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod
- Create a Role named "foo" with subresource permissions:
  - kubectl create role foo --verb=get,list,watch --resource=pods,pods/status
- Create a Role named "foo" with apiGroups specified:
  - kubectl create role foo --verb=get,list,watch --resource=replicasets.apps
- Get Help
  - kubectl create role -h

Create RoleBinding Using kubectl Examples

- Within the namespace "acme", grant the permissions in the "admin" ClusterRole to a user named "bob":
  - kubectl create rolebinding bob-admin-binding --clusterrole=admin --user=bob --namespace=acme
- Within the namespace "acme", grant the permissions in the "view" ClusterRole to the service account in the namespace "acme" named "myapp":
  - kubectl create rolebinding myapp-view-binding --clusterrole=view --serviceaccount=**acme:myapp** --namespace=acme
- Within the namespace "acme", grant the permissions in the "view" ClusterRole to a service account in the namespace "myappnamespace" named "myapp":
  - kubectl create rolebinding myappnamespace-myapp-view-binding --clusterrole=view --serviceaccount=myappnamespace:myapp --namespace=acme

Role and RoleBinding YAML Example

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

role-pod-reader.yml

```yaml
- kind: Group
  name: <GROUP NAME>
  apiGroup: rbac.authorization.k8s.io
```

```yaml
- kind: ServiceAccount
  name: <Service Account NAME>
  apiGroup: ""
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the
"default" namespace.
# You need to already have a Role named "pod-reader" in that
namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
  name: anurak # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or
ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
```

rolebinding-read-pods.yml

ClusterRole and ClusterRoleBinding

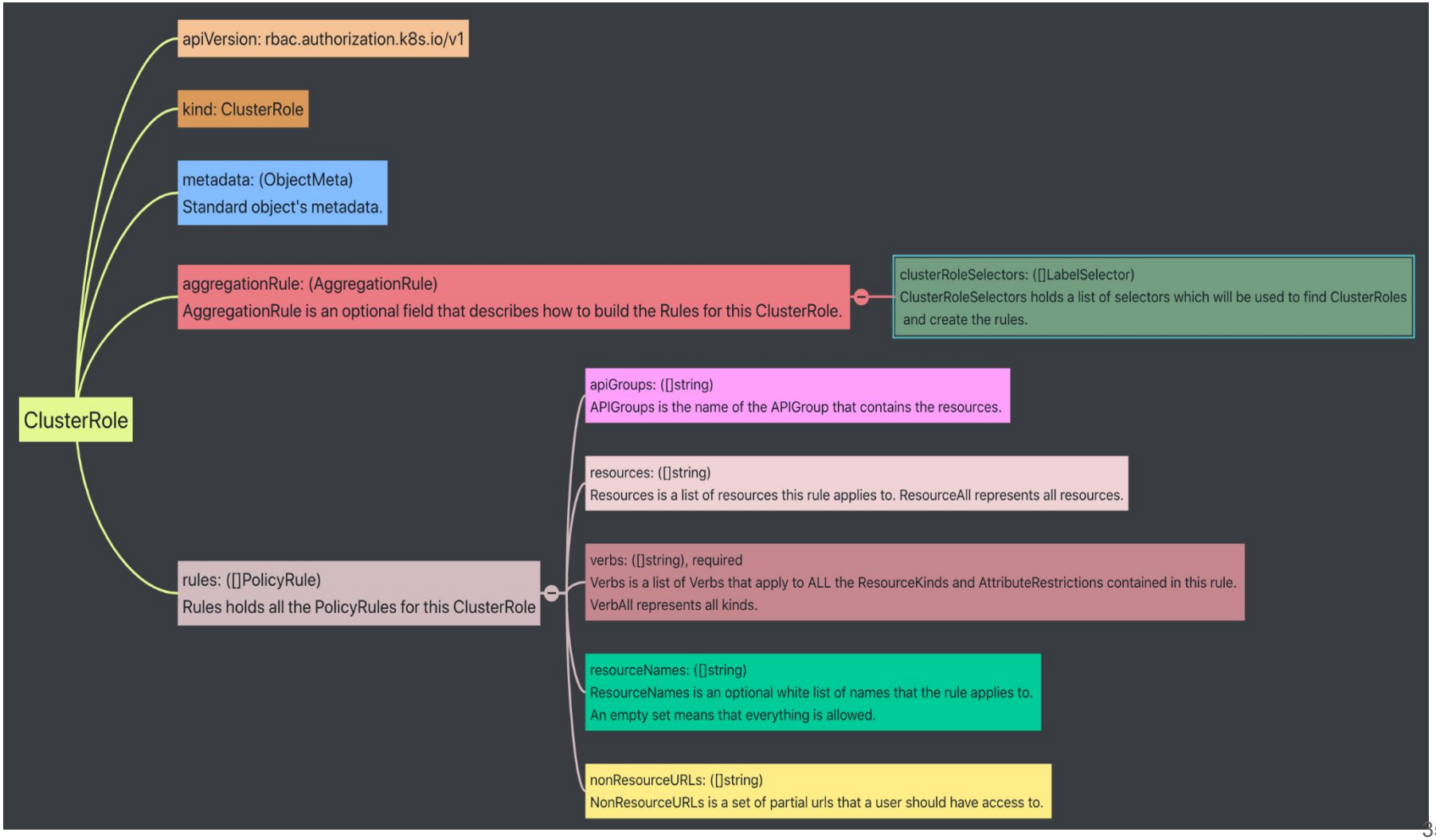
- ClusterRole and ClusterRoleBinding, by contrast, are non-namespaced resources.
- They are used to grant permissions to cluster-wide resources
- Use ClusterRole to grant access to:
  - cluster-scoped resources (like nodes)
  - non-resource endpoints (like /healthz)
  - namespaced resources (like Pods), across all namespaces
- You can aggregate several ClusterRoles into one combined ClusterRole.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac.example.com/aggregate-to-monitoring: "true"
rules: [] # The control plane automatically fills in the rules
```

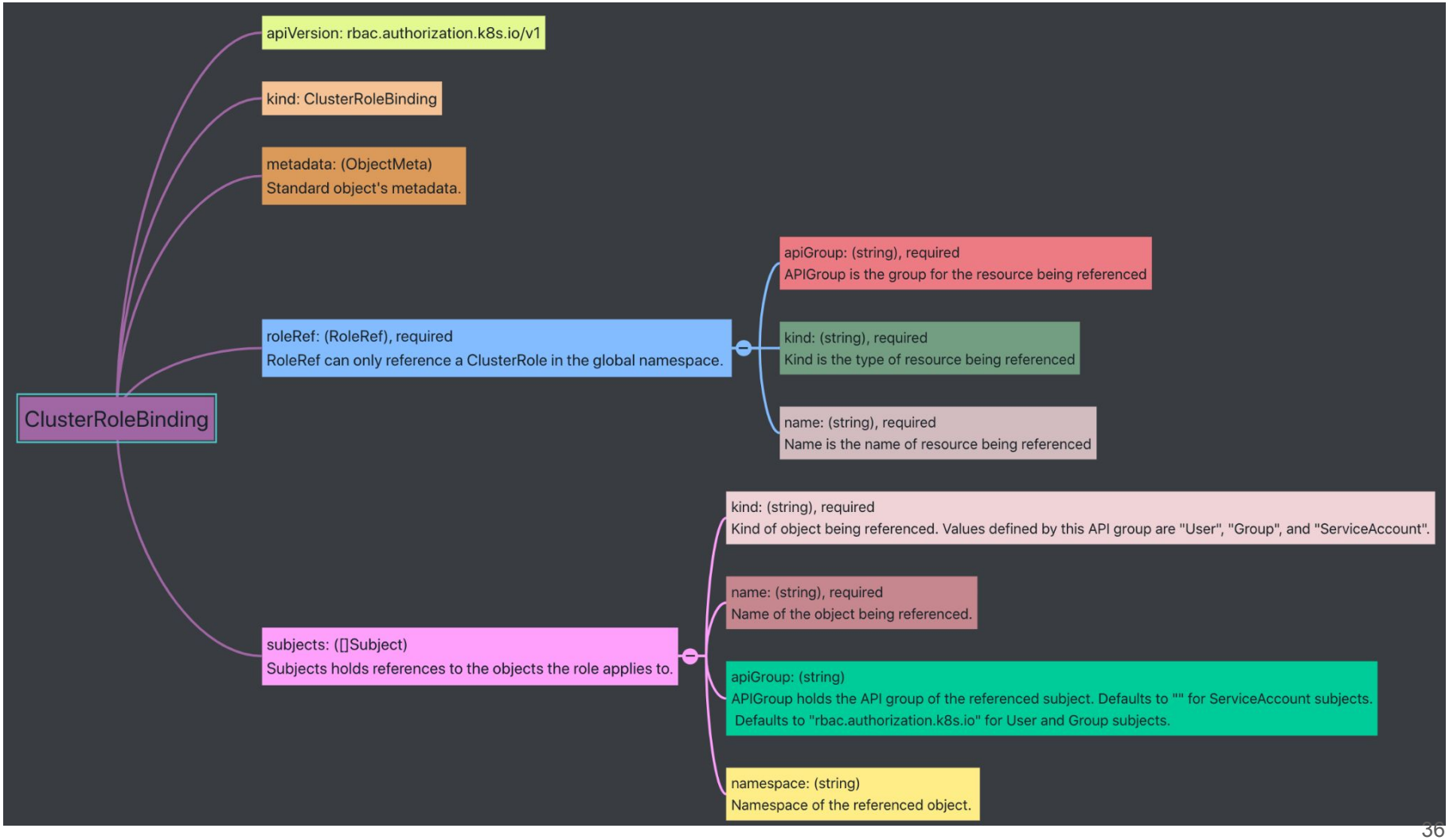
Default ClusterRole and ClusterRoleBinding

- API servers create a set of default ClusterRole and ClusterRoleBinding objects.
- Prefix with **“system:”**
- All of the default ClusterRoles and ClusterRoleBindings are labeled with kubernetes.io/bootstrapping=rbac-defaults.
- kubectl get clusterrole -l kubernetes.io/bootstrapping=rbac-defaults
- Some of the default ClusterRoles are not **system:** prefixed.
  - cluster-admin - Allows super-user access to perform any action on any resource.
  - admin - Allows admin access, intended to be granted within a namespace using a RoleBinding.
  - edit - Allows read/write access to most objects in a namespace. This role does not allow viewing or modifying roles or role bindings.
  - view - Allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings.
  - kubectl get clusterrole cluster-admin -oyaml
  - kubectl get clusterrolebinding cluster-admin -oyaml

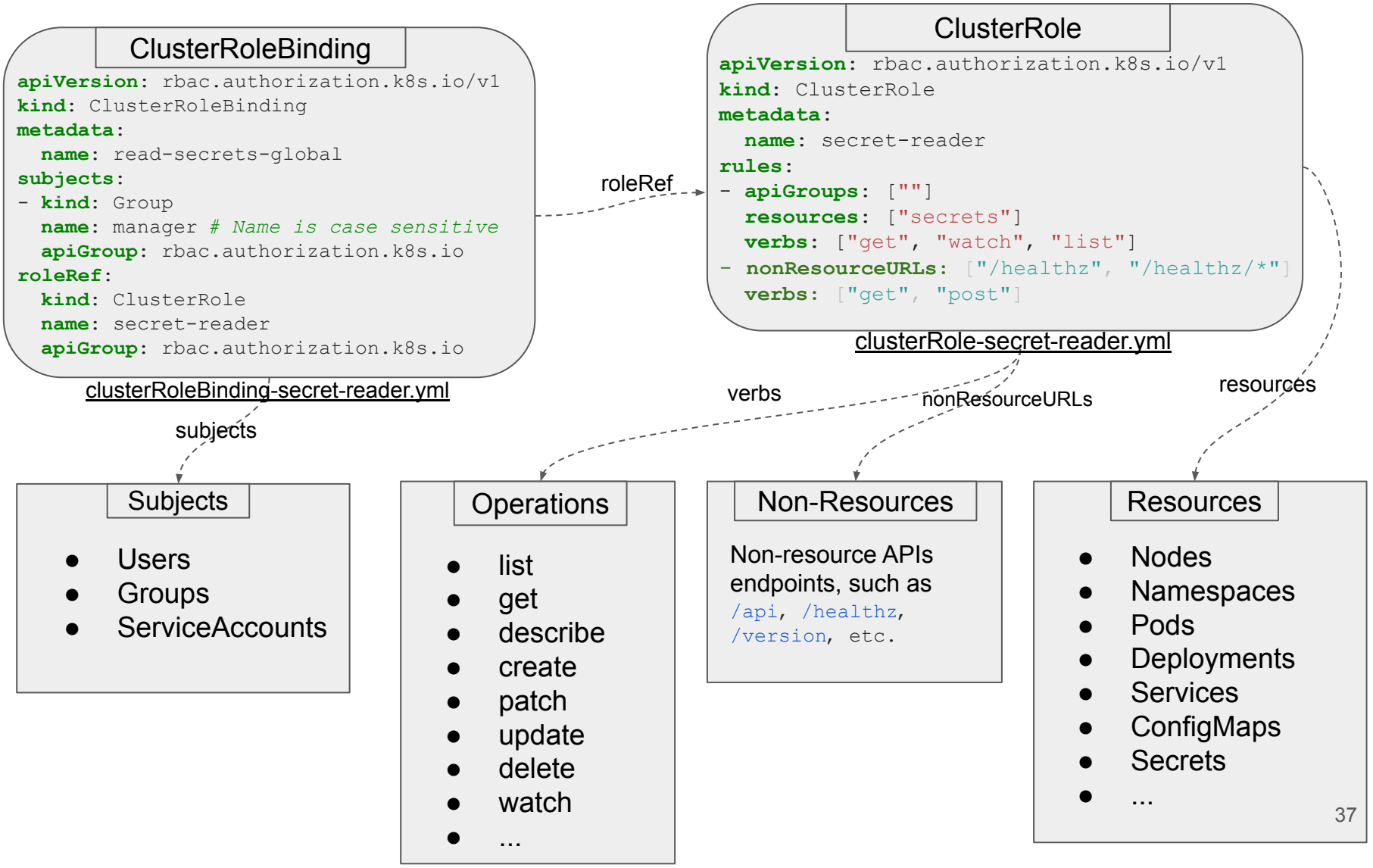
ClusterRole Object



ClusterRoleBinding Object



RBAC ClusterRoles and ClusterRoleBinding



ClusterRole Examples

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 # "namespace" omitted since ClusterRoles are not
namespaced
 name: secret-reader
rules:
- apiGroups: [""]
 resources: ["secrets"]
 verbs: ["get", "watch", "list"]
- nonResourceURLs: ["/healthz", "/healthz/*"]
 verbs: ["get", "post"]
```

clusterRole-secret-reader.yml

- kubectl apply -f clusterRole-secret-reader.yml
- kubectl create rolebinding secret-reader --clusterrole=secret-reader --serviceaccount default:anuraksa --namespace=default
- Within busybox shell
  - curl -H "Authorization: Bearer $TOKEN" https://kubernetes/healthz?verbose --insecure

Create ClusterRole Using kubectl Exaples

- Create a ClusterRole named "pod-reader" that allows user to perform get, watch and list on pods:
  - kubectl create clusterrole pod-reader --verb=get,list,watch --resource=pods
- Create a ClusterRole named "pod-reader" with resourceNames specified:
  - kubectl create clusterrole pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod
- Create a ClusterRole named "foo" with apiGroups specified:
  - kubectl create clusterrole foo --verb=get,list,watch --resource=replicasets.apps
- Create a ClusterRole named "foo" with nonResourceURL specified:
  - kubectl create clusterrole "foo" --verb=get --non-resource-url=/logs/*
- Create a ClusterRole named "monitoring" with an aggregationRule specified:
  - kubectl create clusterrole monitoring --aggregation-rule="rbac.example.com/aggregate-to-monitoring=true"

Create ClusterRoleBinding Using kubectl Examples

- Across the entire cluster, grant the permissions in the "cluster-admin" ClusterRole to a user named "root":
  - kubectl create clusterrolebinding root-cluster-admin-binding --clusterrole=cluster-admin --user=root
- Across the entire cluster, grant the permissions in the "system:node-proxier" ClusterRole to a user named "system:kube-proxy":
  - kubectl create clusterrolebinding kube-proxy-binding --clusterrole=system:node-proxier --user=system:kube-proxy
- Across the entire cluster, grant the permissions in the "view" ClusterRole to a service account named "myapp" in the namespace "acme":
  - kubectl create clusterrolebinding myapp-view-binding --clusterrole=view --serviceaccount=acme:myapp

End.