Part 5:

Scenario:

Three teams with different access needs:

1. Developers: Deploy and manage applications in dev namespace

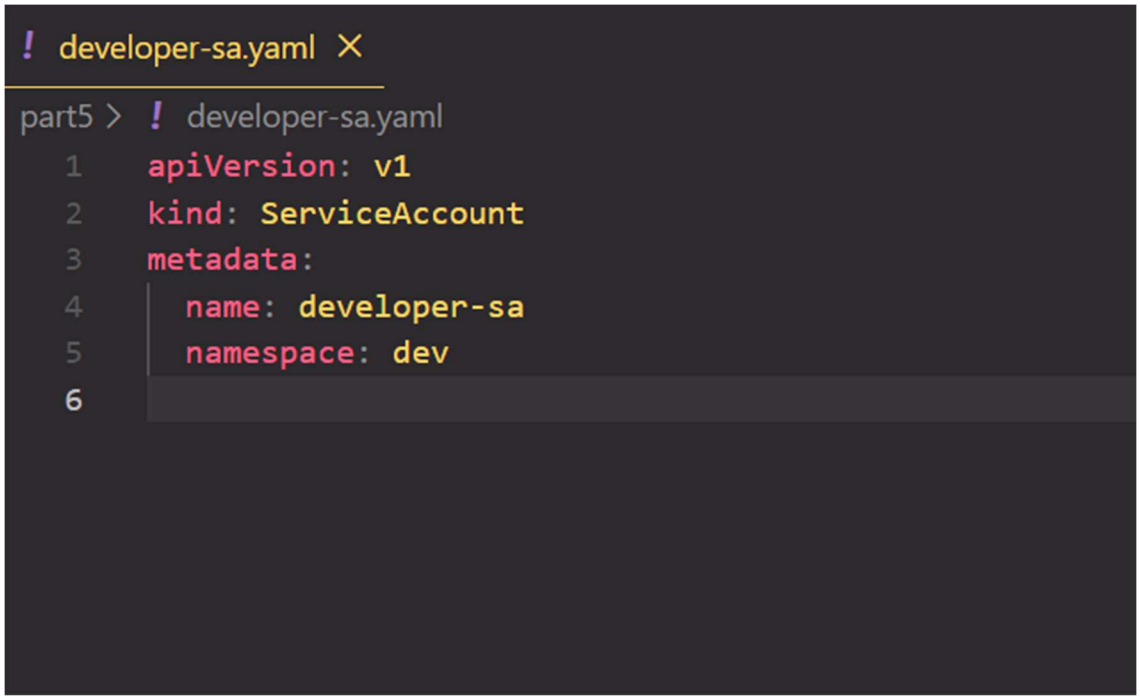
2. QA Team: Read-only access to dev and staging namespaces

3. Ops Team: Cluster-wide read access + write access to production namespace only

Complete RBAC Design:

1. Developers Team

Service account:



Role (Namespace-scoped):

part5 > ! developer-role.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer-role
  namespace: dev
rules:
  # Full access to application resources
  - apiGroups: ["", "apps", "batch"]
    resources:
      - pods
      - pods/log
      - pods/exec
      - deployments
      - replicasets
      - services
      - configmaps
      - jobs
      - cronjobs
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]

  # Read-only access to resource quotas and limits
  - apiGroups: [""]
    resources: ["resourcequotas", "limitranges"]
    verbs: ["get", "list"]
```

Role Binding:

part5 > ! developer-binding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: developer-binding
  namespace: dev
subjects:
  - kind: ServiceAccount
    name: developer-sa
    namespace: dev
  # Can also bind to Groups for user authentication
  - kind: Group
    name: developers
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer-role
  apiGroup: rbac.authorization.k8s.io
```

Justification for Role (not ClusterRole):

- Developers only need access within the dev namespace
- Using a Role limits blast radius if credentials are compromised
- Follows least privilege. Therefore, there is no reason to grant cluster-wide permissions
- Easier to audit and manage namespace-specific permissions

2. QA Team

Service account:

part5 > ! qa-sa.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: qa-sa
  namespace: default # Can be in any namespace

```

ClusterRole (Reusable read-only role):

part5 > ! qa-reader-clusterrole.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: qa-reader-clusterrole
rules:
  # Read-only access to application resources
  - apiGroups: ["", "apps", "batch"]
    resources:
      - pods
      - pods/log
      - deployments
      - replicasets
      - services
      - configmaps
      - jobs
      - cronjobs
    verbs: ["get", "list", "watch"]

  # Read events for troubleshooting
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["get", "list", "watch"]

```

RoleBinding for dev namespace:

part5 > ! qa-reader-dev-binding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: qa-reader-dev-binding
  namespace: dev
subjects:
  - kind: ServiceAccount
    name: qa-sa
    namespace: default
  - kind: Group
    name: qa-team
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole # Binding to a ClusterRole
  name: qa-reader-clusterrole
  apiGroup: rbac.authorization.k8s.io

```

RoleBinding for staging namespace:

part5 > ! qa-reader-staging-binding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: qa-reader-staging-binding
  namespace: staging
subjects:
  - kind: ServiceAccount
    name: qa-sa
    namespace: default
  - kind: Group
    name: qa-team
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: qa-reader-clusterrole
  apiGroup: rbac.authorization.k8s.io

```

Justification for ClusterRole + RoleBindings:

- **ClusterRole:** Creates a reusable read-only permission template
- **RoleBindings:** Limits the ClusterRole's scope to specific namespaces (dev and staging only)
- **Why not a simple Role?:** ClusterRole can be reused across multiple namespaces with different RoleBindings
- **Why not ClusterRoleBinding?:** That would grant access to ALL namespaces, including production which may arise security risk.
- **Best of both worlds:** Reusability + namespace isolation

3. OPs Team

Service account:

part5 > ! ops-sa.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ops-sa
  namespace: default

```

ClusterRole for cluster-wide read:

part5 > ! ops-global-reader.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ops-global-reader
rules:
  # Read all resources cluster-wide
  - apiGroups: ["", "apps", "batch", "networking.k8s.io", "storage.k8s.io"]
    resources:
      - pods
      - deployments
      - services
      - nodes
      - namespaces
      - persistentvolumes
      - persistentvolumeclaims
      - ingresses
      - events
    verbs: ["get", "list", "watch"]

  # Read cluster-level resources
  - apiGroups: [""]
    resources: ["nodes", "persistentvolumes"]
    verbs: ["get", "list", "watch"]

```

ClusterRoleBinding for global read:

part5 > ! ops-global-reader-binding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ops-global-reader-binding
subjects:
  - kind: ServiceAccount
    name: ops-sa
    namespace: default
  - kind: Group
    name: ops-team
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: ops-global-reader
  apiGroup: rbac.authorization.k8s.io
```

Role for production write access:

part5 > ! ops-production-admin.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ops-production-admin
  namespace: production
rules:
  # Full access to production namespace
  - apiGroups: ["", "apps", "batch", "networking.k8s.io"]
    resources:
      - pods
      - deployments
      - replicasets
      - services
      - configmaps
      - secrets
      - jobs
      - cronjobs
      - ingresses
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]

  # Access to pod exec and logs for debugging
  - apiGroups: [""]
    resources: ["pods/exec", "pods/log"]
    verbs: ["get", "create"]
```

RoleBinding for production:

part5 > ! ops-production-admin-binding.yaml

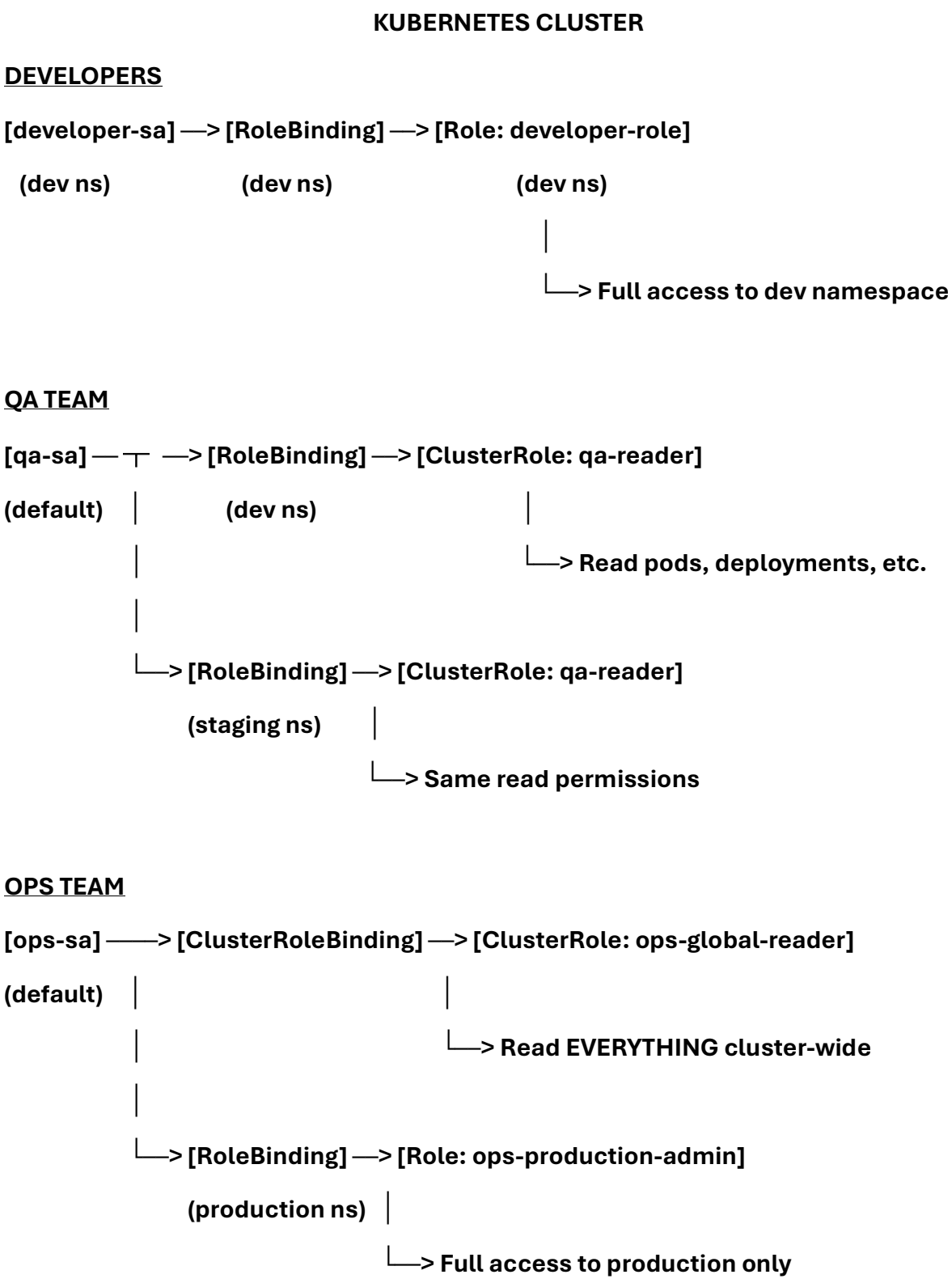
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ops-production-admin-binding
  namespace: production
subjects:
  - kind: ServiceAccount
    name: ops-sa
    namespace: default
  - kind: Group
    name: ops-team
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: ops-production-admin
  apiGroup: rbac.authorization.k8s.io

```

Justification for Mixed Approach:

- **ClusterRole + ClusterRoleBinding:** Needed for cluster-wide read access (nodes, all namespaces overview)
- **Role + RoleBinding:** Limits write access to production namespace only
- **Why both?:** Ops needs visibility across the entire cluster but should only modify production
- **Security:** Prevents accidental changes to dev/staging while troubleshooting
- **Separation of concerns:** Read permissions are global, write permissions are targeted

Visual relationship Diagram:



Summary Table

Team	ServiceAccount	Role/ClusterRole	Binding Type	Scope	Justification
Developers	developer-sa (dev ns)	Role: developer-role	RoleBinding (dev)	dev namespace only	Namespace-scoped prevents access to other environments
QA Team	qa-sa (default ns)	ClusterRole: qa-reader-clusterrole	2x RoleBindings (dev, staging)	dev + staging namespaces	ClusterRole for reusability, RoleBindings for namespace isolation
Ops Team	ops-sa (default ns)	ClusterRole: ops-global-reader + Role: ops-production-admin	ClusterRoleBinding (read) + RoleBinding (production write)	Cluster-wide read + production write	Mixed approach: global visibility, targeted write access