# Online Talk: RBAC in Kubernetes

https://github.com/javsalgar/rbac-online-talk

kubernetes

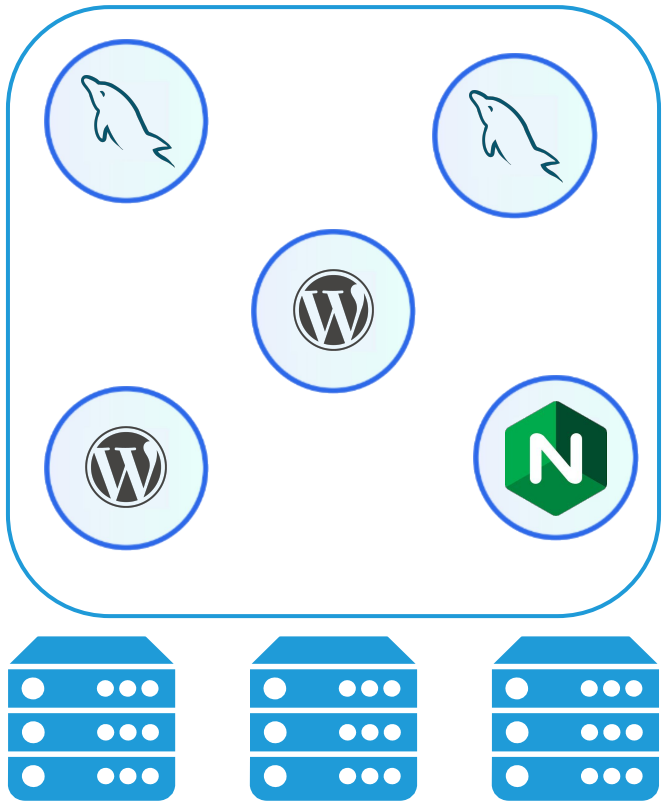
bitnami

# I - Creating users




bitnami

# Question



- When starting with K8s, we tend to use full administrator credentials. Examples: minikube, k8s sandbox...
- In a real cluster we may want to have different users, groups and privileges

Developer

user: jsalmeron
group: dev, tech-lead

Administrator

user: jjo
group: sre, tech-lead

Developer

user: juan
group: dev

Administrator

user: dbarranco
group: sre, devops

- If in Kubernetes everything is modelled as an API Object, maybe there's something like

~~`kubectl create user ...`~~

# User management in Kubernetes


- Kubernetes provides no API objects for users*
- User management must be configured by the cluster administrator. Examples:
  - Certificate-based authentication
  - Token-based authentication
  - Basic authentication
  - OAuth2

*At least something like we have for Deployments, Pods… etc.

# Certificate-based authentication


- Kubernetes is configured with a Certificate Authority (CA)

`/etc/kubernetes/pki/ca.crt`

Public certificate

`/etc/kubernetes/pki/ca.key`

Private key

- Every SSL certificate signed with this CA will be accepted by the Kubernetes API
- Possible options for creating certificates: OpenSSL or CloudFlare's PKI toolkit
- Two important fields in the SSL certificate:
  - Common Name (CN): Kubernetes will interpret this value as the **user**
  - Organization (O): Kubernetes will interpret this value as the **group**

# Creating user certificate: steps



- Create private key (if it does not exist)

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

- Create certificate signing request (CSR)

```
openssl req -new -key juan.key -out juan.csr -subj "/CN=juan/O=devs"
```

user group

- Send the CSR to the administrator


- Create certificate from CSR using the cluster authority 

```
openssl x509 -req -in juan.csr -CA CA_LOCATION/ca.crt -CAkey
CA_LOCATION/ca.key -CAcreateserial -out juan.crt -days 500
```

# Next step: Create kubectl configuration


- To add in your local machine the new configuration:
  - Download the cluster authority and generated certificate
  - Add the new cluster to kubectl

```
kubectl config set-cluster sandbox --certificate-authority=ca.pem
--embed-certs=true --server=https://<PUBLIC_ADDRESS_OF_YOUR_CLUSTER>:6443
```

  - Add the new credentials to kubectl

```
kubectl config set-credentials juan --client-certificate=juan.crt
--client-key=juan.key --embed-certs=true
```

  - Add the new context to kubectl

```
kubectl config set-context sandbox-juan --cluster=sandbox --user=juan
```

# Finally: Test your new configuration


- Change to the newly created context

```
kubectl config use-context sandbox-juan
```

You can have multiple clusters and configurations

- Let's execute a basic command

```
kubectl get pods
```

- What happened?

```
Error from server (Forbidden): pods is forbidden: User "juan" cannot list
pods in the namespace "default"
```

# II - Role Based Access Control (RBAC)



bitnami

# RBAC in Kubernetes

- Three important groups


**Subjects**

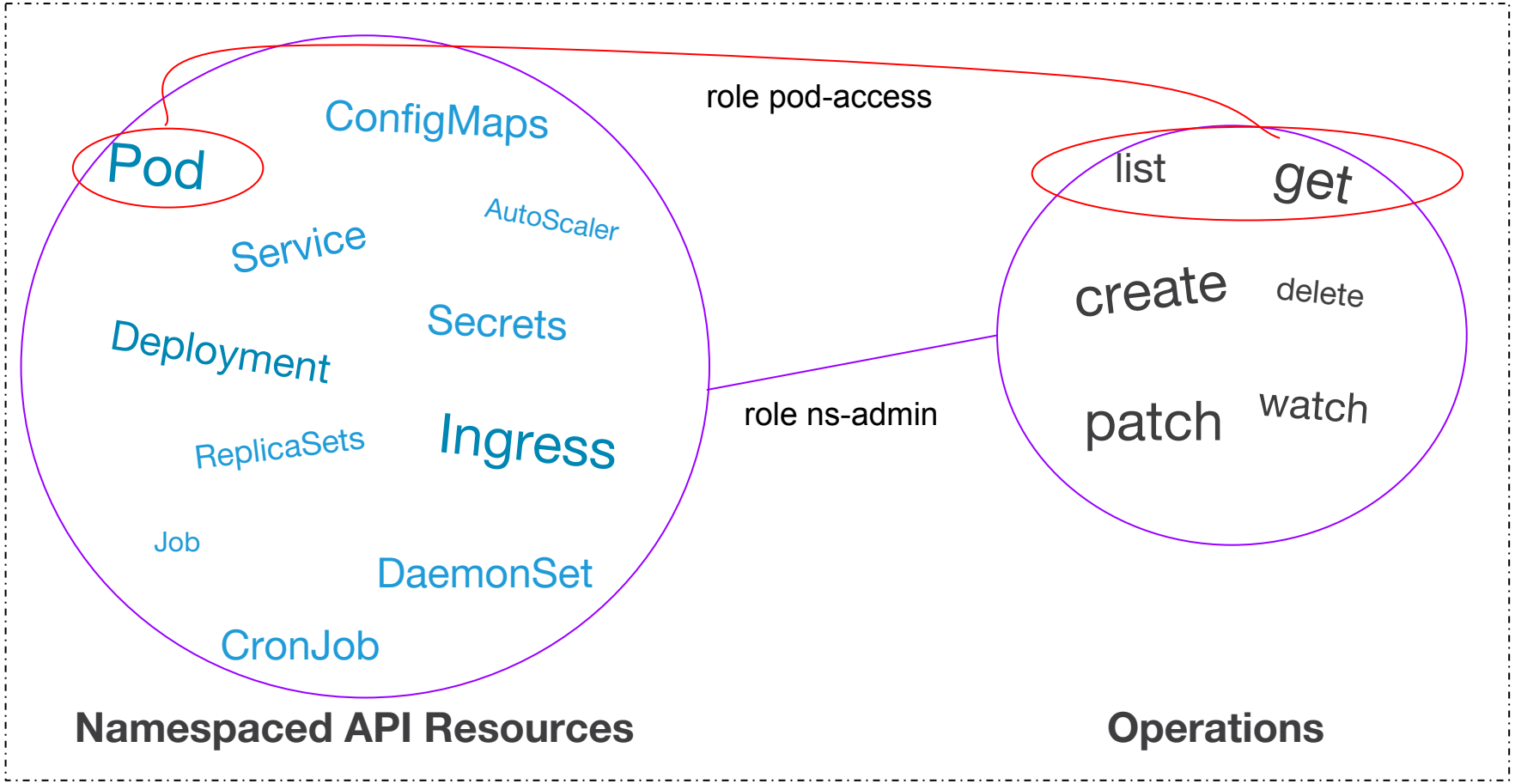

**API Resources**


**Operations (Verbs)**

- RBAC connects the three of them

# RBAC in Kubernetes: Roles

- Establish a set of allowed operations (rules) over a set of resources **in a namespace**




**namespace "test"**

# RBAC in Kubernetes: Roles

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   namespace: test
   name: pod-access
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list"]
```

WHICH RESOURCES

WHICH OPERATIONS

- Need to specify:
  - Api group
  - Name

Find it in the API reference, examples

| Group | Version | Kind |
|---|---|---|
| apps | v1 | Deployment |

| Group | Version | Kind |
|---|---|---|
| core | v1 | Pod |

When it is core, we use an empty string

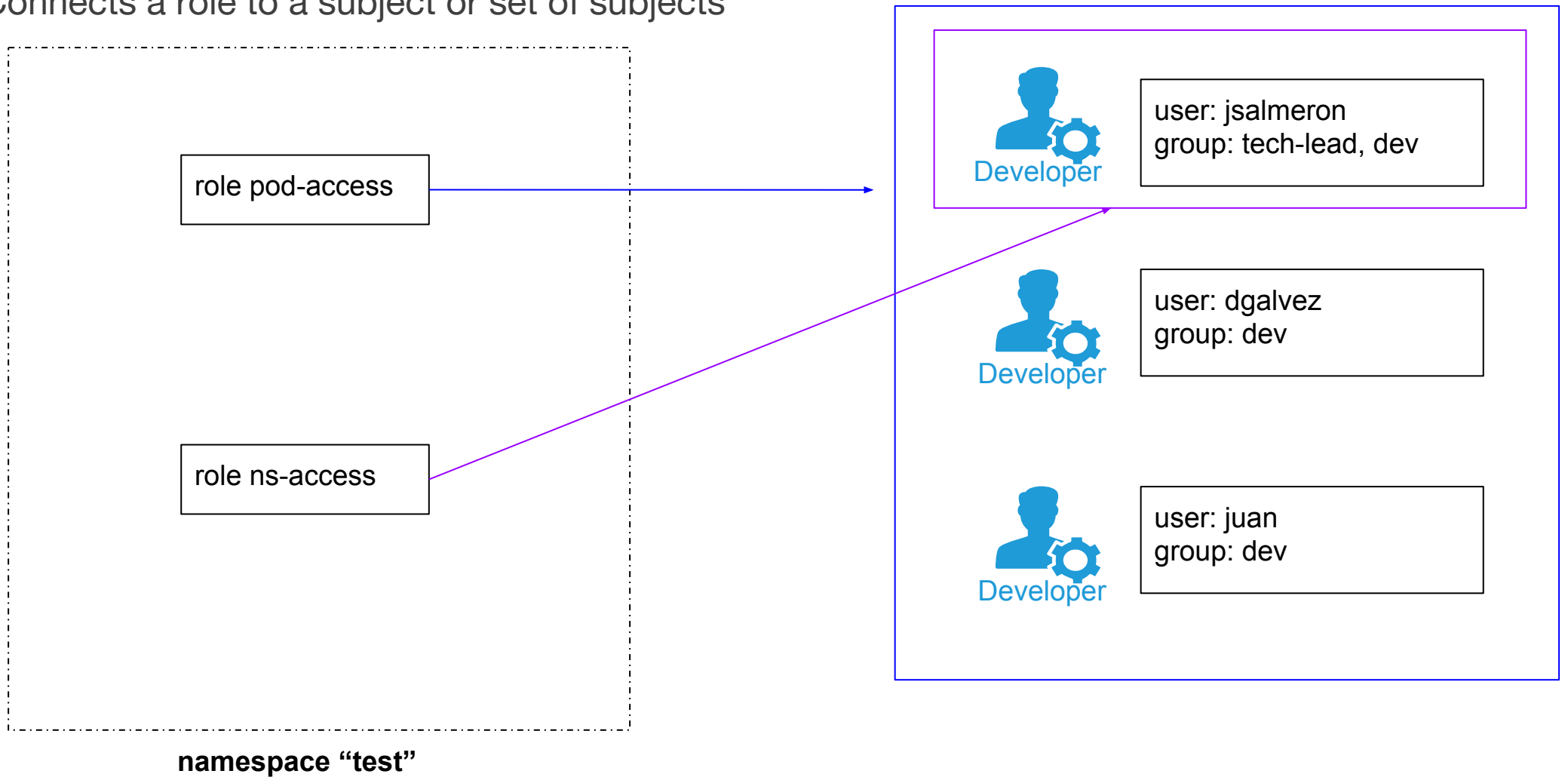
# RBAC in Kubernetes: Roles

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   namespace: test
   name: ns-admin
rules:
 - apiGroups: ["*"]
   resources: ["*"]
   verbs: ["*"]
```

- Wildcards are allowed

# RBAC in Kubernetes: RoleBindings

- Connects a role to a subject or set of subjects



# RBAC in Kubernetes: RoleBinding

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: devs-read-pods
  namespace: test
subjects:
- kind: Group
  name: devs
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-access
  apiGroup: rbac.authorization.k8s.io
```

- Examples
  - User
  - Group
  - ...

Later we will see another one

WHICH SUBJECTS

Used to specify which api group the kind belongs to

WHICH ROLE (ONLY ONE PER BINDING)

# RBAC in Kubernetes: RoleBinding

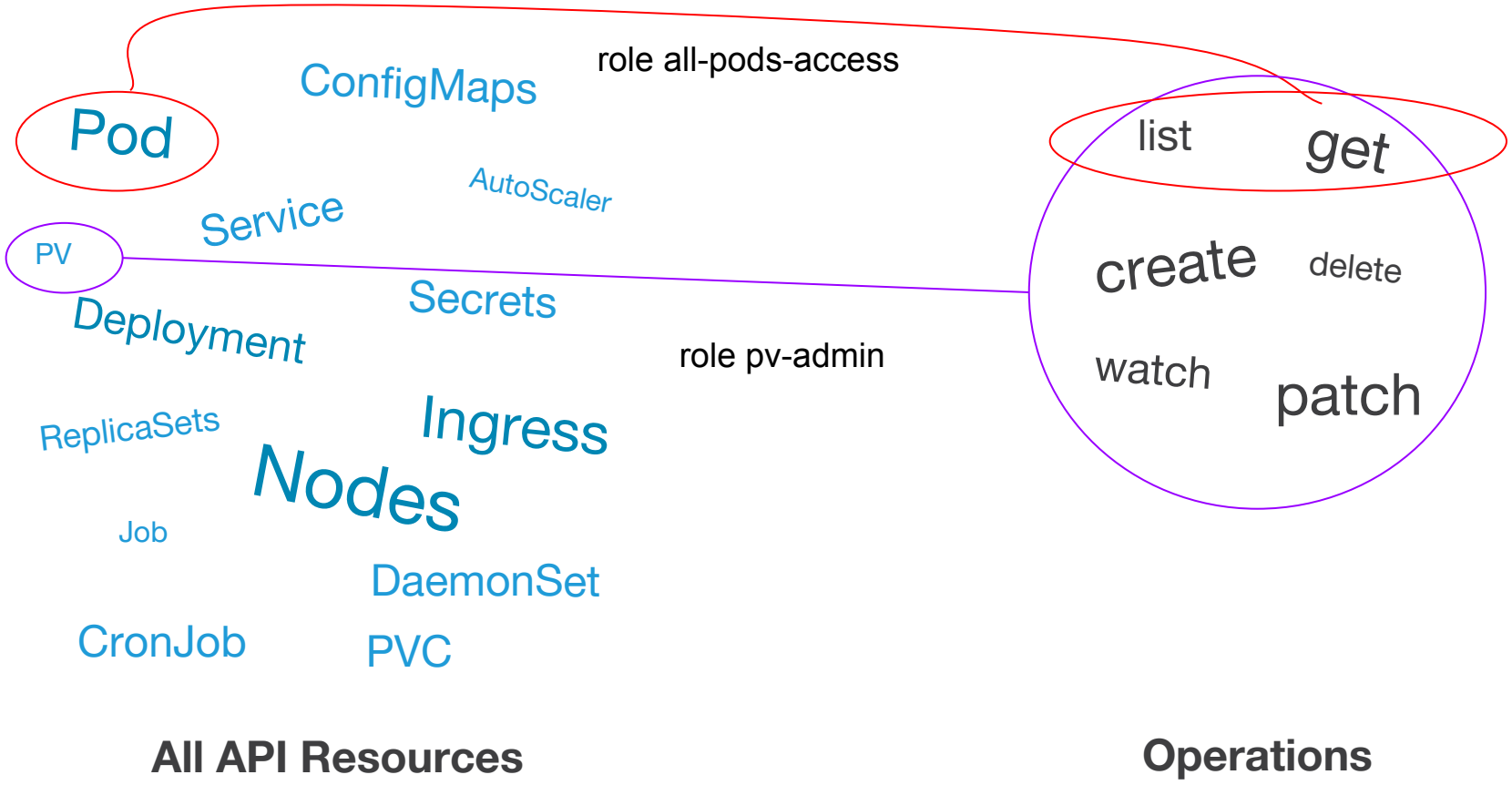
```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: salme-ns-admin
  namespace: test
subjects:
- kind: User
  name: jsalmeron # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: ns-admin
  apiGroup: rbac.authorization.k8s.io
```



Mini-exercise: Another way of doing this?

# RBAC in Kubernetes: ClusterRoles

- Establish a set of allowed operations over a set of resources in the whole cluster



# RBAC in Kubernetes: ClusterRoles

- Roles and ClusterRoles have very similar yaml

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   name: pod-access
   namespace: test
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list"]
```

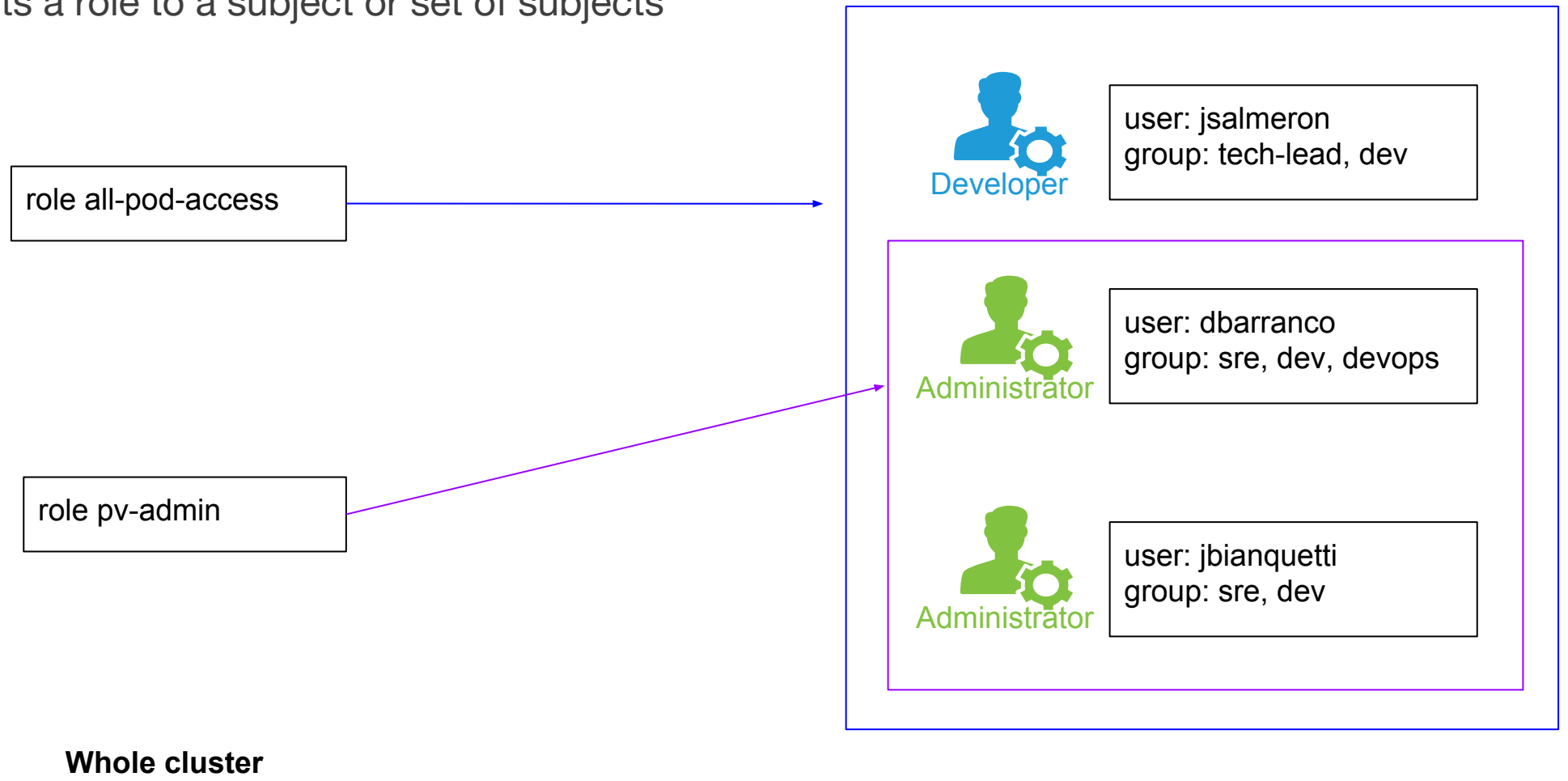
Only difference

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   name: all-pod-access

rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list"]
```

# RBAC in Kubernetes: ClusterRoleBinding

- Connects a role to a subject or set of subjects



# RBAC in Kubernetes: ClusterRoleBinding

- Just like the previous case, very similar YAML

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: devs-read-pods
  namespace: test
subjects:
- kind: User
  name: jsalmeron # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: ns-admin
  apiGroup: rbac.authorization.k8s.io
```

Only differences

Only differences

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: salme-reads-all-pods

subjects:
- kind: User
  name: jsalmeron # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: all-pod-access
  apiGroup: rbac.authorization.k8s.io
```

# Default ClusterRoleBindings

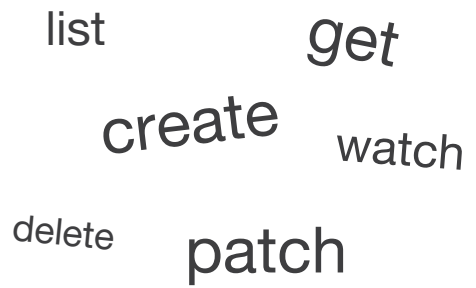
- Kubernetes includes some ClusterRoleBindings. For example:
  - **system:basic-user**: For unauthenticated users (group **system:unauthenticated**). No operations are allowed.
  - **cluster-admin:** For members of the **system:masters** group. Can do any operation on the cluster (using **cluster-admin** ClusterRole).


Admin accounts can be created belonging to this group

```
openssl req ... -subj "/CN=dbarranco/O=system:masters"
```

  - ClusterRoleBindings for the **different components** of the cluster (kube-controller-manager, kube-scheduler, kube-proxy …)

# More about the possible actions (verbs)




- TRIVIA: Example operations and their requirements

| | |
|---|---|
| `kubectl run --image=bitnami/mongodb my-mongodb` | deployments: create |
| `kubectl get deployments -w` | deployments: get, list, watch |
| `kubectl delete deployment my-mongodb` | deployments: get, delete |
| `kubectl edit deployment my-mongodb mypod` | deployments: get, patch |
| `kubectl expose deployment my-mongodb --port=27017 --type=NodePort` | deployments: get<br>services: create |
| `kubectl exec -ti mypod bash` | pods: get<br>pods/exec: create |

# Questions

- Find the necessary RBAC rules so the user can contact Helm's Tiller pod
- We know that this command should work with the previously created RBAC rules (salme-ns-admin)

```
helm install stable/wordpress --namespace test
```

- And what about this command?

```
helm install stable/wordpress --namespace default
```

- Regenerate the Tiller pod and try the command again

```
helm reset --force && helm init
```

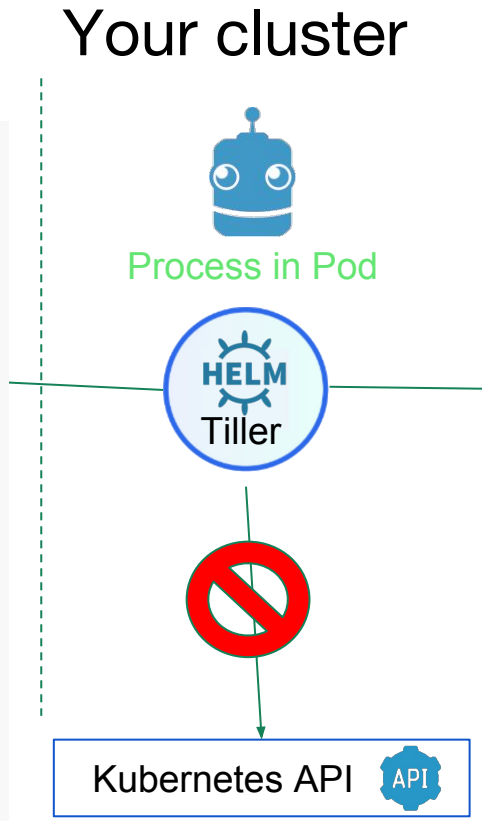
```
Error: rpc error: code = Unknown desc = configmaps is forbidden: User
"system:serviceaccount:kube-system:default" cannot list configmaps in the namespace
"kube-system"
```

# Helm under the hood

A server called tiller is in charge of rendering and deploying charts

```
helm install my-wordpress/
```

```
kind: Deployment
metadata:
  name: {{ template "fullname" . }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    spec:
      containers:
      - name: wp
        image: {{ .Values.image }}
...
```

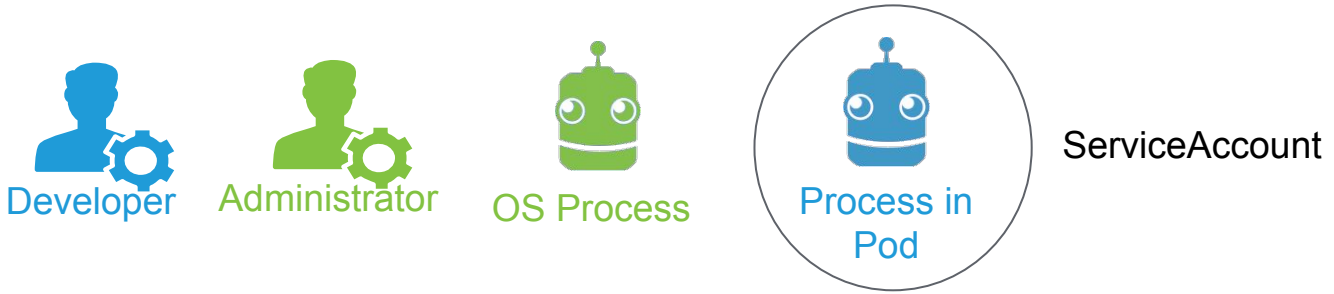



```
kind: Deployment
metadata:
  name: pilfering-anaconda
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: wp
        image: bitnami/wordpress:4.8.3
...
```

How do we configure this? Do we need to provide a certificate to the pod?

# RBAC in Kubernetes (again): ServiceAccount


- While regular users are not handled by Kubernetes, processes inside pods do have an API object




- Necessary for **pods** that need to contact Kubernetes API
- Also used for other operations like storing image pull secrets

# RBAC in Kubernetes (again): ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
```

- An API token will be automatically created and stored in the cluster

- Can be used in RoleBinding and ClusterRoleBinding as subjects
- ServiceAccounts are used in Pod/RS/Deployment declarations

- Examples
  - User
  - Group
  - ...

Later we will see another one

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: my-service-account
```

- The API token will be mounted inside the containers

If not specified it will use the “default” ServiceAccount

# Deploying Tiller

- Create a Tiller ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller-sa
  namespace: kube-system
```

- Set up RBAC for Tiller
  - Which operations requires Tiller?
  - In principle, it can deploy ANYTHING in ANY NAMESPACE

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tiller-rolebinding

subjects:
- kind: ServiceAccount
  name: tiller-sa # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

# Deploying Tiller

- Update the tiller pod

```
helm init --service-account tiller-sa --upgrade
```

- Let's check if Tiller works now

```
helm ls
```

# Next steps in Kubernetes Cluster Administration

- Different type of authentications like OAuth
- Limits and Quotas: ResourceQuota and LimitRanges
- NetworkPolicies
- PodSecurityPolicies

Check Bitnami Documentation for several Kubernetes How-To's:

https://docs.bitnami.com/kubernetes/how-to/

# Thank You

For more information, visit bitnami.com