AUGUST 2023

IT & DATA SCIENCE

Security Best Practices for AI infrastructure – RBAC

Table of contents

Introduction

In the ever-evolving landscape of data science, containers have emerged as an indispensable tool, revolutionizing how data scientists work. The adoption of containers has skyrocketed, and for good reason – many popular data science tools are now built and optimized for containerization. TensorFlow, PyTorch, Keras, and a plethora of pre-trained models are just a few examples of some of the powerful tools available, within containers that streamline the data science process.

One name that stands out in the world of container orchestration is Kubernetes. As the de facto standard for managing containers, Kubernetes has become an essential part of data scientists' toolsets. In 2021, a report from Run:ai discovered that 42% of respondents said they used Kubernetes for AI/ML workflows. And, last year Red Hat found that that number had increased to 65%, with this year expected to be even higher.

However, while empowering data scientists, security concerns can arise. So, how can we ensure that containers and Kubernetes are used in a way that doesn't compromise the integrity of the system or the privacy of data? The security team recognizes that researchers should be restricted to their designated containers without any ability to tamper with system components or interfere with other researchers' workspaces. Unauthorized access to sensitive data and shared resources must be prevented as well.

Introducing Kuberenetes RBAC - Kubernetes Role-Based Access Control (RBAC) is a security mechanism that enables administrators to define and manage fine-grained access permissions for users or groups within a Kubernetes cluster. RBAC allows for the assignment of specific roles to users, which determines their access rights to resources and operations within the cluster. By implementing RBAC, administrators can ensure that only authorized personnel have the appropriate privileges to perform certain actions, reducing the risk of unauthorized access and potential security breaches. This helps in maintaining a secure and controlled environment, ensuring that each user has precisely the necessary permissions to perform their tasks while restricting access to sensitive resources, ultimately enhancing the overall security and stability of the Kubernetes cluster.

How do we use it?

Let's say we have 2 data scientists that need to share the resources on the cluster, Bob and Alice, but we want to make sure that they can't access each other's containers and data.
These are the steps you need to take to ensure that Bob and Alice can run their experiments on the cluster, each having access only to his containers and data:

1. **Create a namespace for each user**

```
kubectl create ns bob
namespace/bob created
```

```
kubectl create ns alice
namespace/alice created
```

2. **Create a service account,** role and role-binding for each user (the service account defines the user, the role defines the scope of actions, and the role binding simply binds the user to the role):
First we will create the sa-role-rolebinding-bob.yml for bob:

```
vi sa-role-rolebinding-bob.yml

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bob-user
  namespace: bob
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bob-user-full-access
  namespace: bob
rules:
  - apiGroups: ["", "extensions", "apps", "batch"]
    resources: ["*"]
    verbs: ["*"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bob-user-view
  namespace: bob
subjects:
  - kind: ServiceAccount
    name: bob-user
    namespace: bob
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: bob-user-full-access
```

3 Create a namespace for each user

```
vi sa-role-rolebinding-alice.yml

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: alice-user
  namespace: alice

---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: alice-user-full-access
  namespace: bob
rules:
  - apiGroups: ["", "extensions", "apps", "batch"]
    resources: ["*"]
    verbs: ["*"]

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: alice-user-view
  namespace: alice
subjects:
  - kind: ServiceAccount
    name: alice-user
    namespace: alice
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: alice-user-full-access
```

```
kubectl apply -f sa-role-rolebinding-alice.yml

serviceaccount/alice-user created
role.rbac.authorization.k8s.io/alice-user-full-access created
rolebinding.rbac.authorization.k8s.io/alice-user-view created
```


```
kubectl apply -f sa-role-rolebinding-alice.yml

serviceaccount/alice-user created
role.rbac.authorization.k8s.io/alice-user-full-access created
rolebinding.rbac.authorization.k8s.io/alice-user-view created
```

4 **Create a 'secret' for your service account (we will need this secret for the next step, to create a user-token for the kubernetes config file):**

```
kubectl -n bob create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: bob-user-sa-token
  annotations:
    kubernetes.io/service-account.name: bob-user
type: kubernetes.io/service-account-token
EOF

secret/bob-user-sa-token created
```

```
kubectl -n alice create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: alice-user-sa-token
  annotations:
    kubernetes.io/service-account.name: alice-user
type: kubernetes.io/service-account-token
EOF

secret/alice-user-sa-token created
```

5 **Create the following environment variables and then the kubernetes config file for each user:**

```
export USER_TOKEN_VALUE=$(kubectl -n bob get secret/bob-user-sa-token -o=go-
template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-template='{{range .contexts}}{{if eq
.name """${CURRENT_CONTEXT}"""}}{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-template='{{range .clusters}}{{if eq .name
"""${CURRENT_CLUSTER}"""}}"{{with index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{
end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-template='{{range .clusters}}{{if eq .name
"""${CURRENT_CLUSTER}"""}}{{ .cluster.server }}{{end}}{{ end }}')
```

Now create the kubernetes config file (the kubernetes config file is the file that each user will use to authenticate to the Kubernetes api server):

```
cat << EOF > bob-kubeconfig
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: bob-user
    namespace: bob
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: bob-user
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

Now, let's do the same for alice:

```
export USER_TOKEN_VALUE=$(kubectl -n alice get secret/alice-user-sa-token -o=go-
template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-template='{{range .contexts}}{{if eq
.name """${CURRENT_CONTEXT}"""}}{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-template='{{range .clusters}}{{if eq .name
"""${CURRENT_CLUSTER}"""}}"{{with index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{
end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-template='{{range .clusters}}{{if eq .name
"""${CURRENT_CLUSTER}"""}}{{ .cluster.server }}{{end}}{{ end }}')
```

Next, we will create the kubernetes config file for alice:

```
cat << EOF > alice-kubeconfig
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: alice-user
    namespace: alice
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: alice-user
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

6 **We are all done! The bob-kubeconfig file should be shared with user bob, and the alice-kubeconfig file should be shared with user alice.**

To verify that it works, lets use the bob-kubeconfig:
Copy the bob-kubeconfig to your .kube directory

```
cp bob-kubeconfig .kube
```

Export the KUBECONFIG environment variable to point to bob-kubeconfig

```
export KUBECONFIG=~/.kube/bob-kubeconfig
```

Now, lets try to list the pods in the bob namespace:

```
kubectl get pods -n bob
No resources found in bob namespace.
```

We were able to list the pods and get a response from the apiserver (that there are no pods in this namespace).
Now, lets try to list the pods in the alice namespace:

```
kubectl get pods -n alice
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:bob:bob-user" cannot
list resource "pods" in API group "" in the namespace "alice"
```

We weren't able to list the pods in the alice namespace, as we only have permissions to the bob namespace!

To summarize, Kubernetes RBAC offers significant benefits to data scientists by providing fine-grained control over access to cluster resources and ensuring data security. With RBAC, data scientists can collaborate seamlessly, streamline operations, and focus on their work without concerns about unauthorized access or potential disruptions.

About Run:ai

Run:ai is an AI management platform for MLOps, Data Science, and DevOps teams. In addition to helping these teams access and utilize their GPU resources more effectively, it also has a powerful set of features that can abstract infrastructure complexities and simplify the process of training and deploying models. With or without a GPU shortage, Run:ai enables data scientists to focus on innovation without having to worry about resource limitations.