SAP Concur

Kubernetes
RBAC

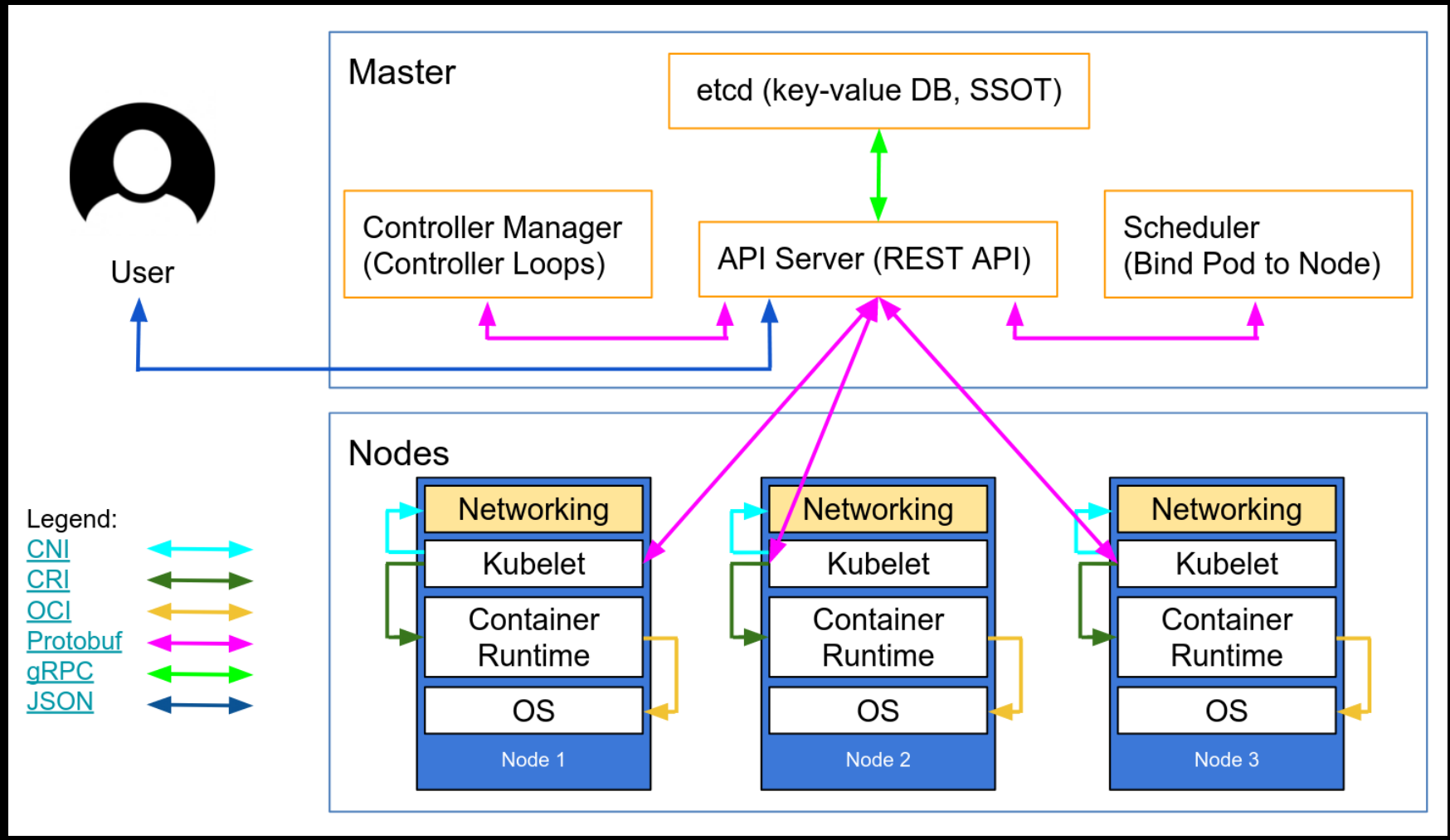
Rafael Troncoso, SAP
December 5, 2019

THE BEST RUN SAP

Disclaimer

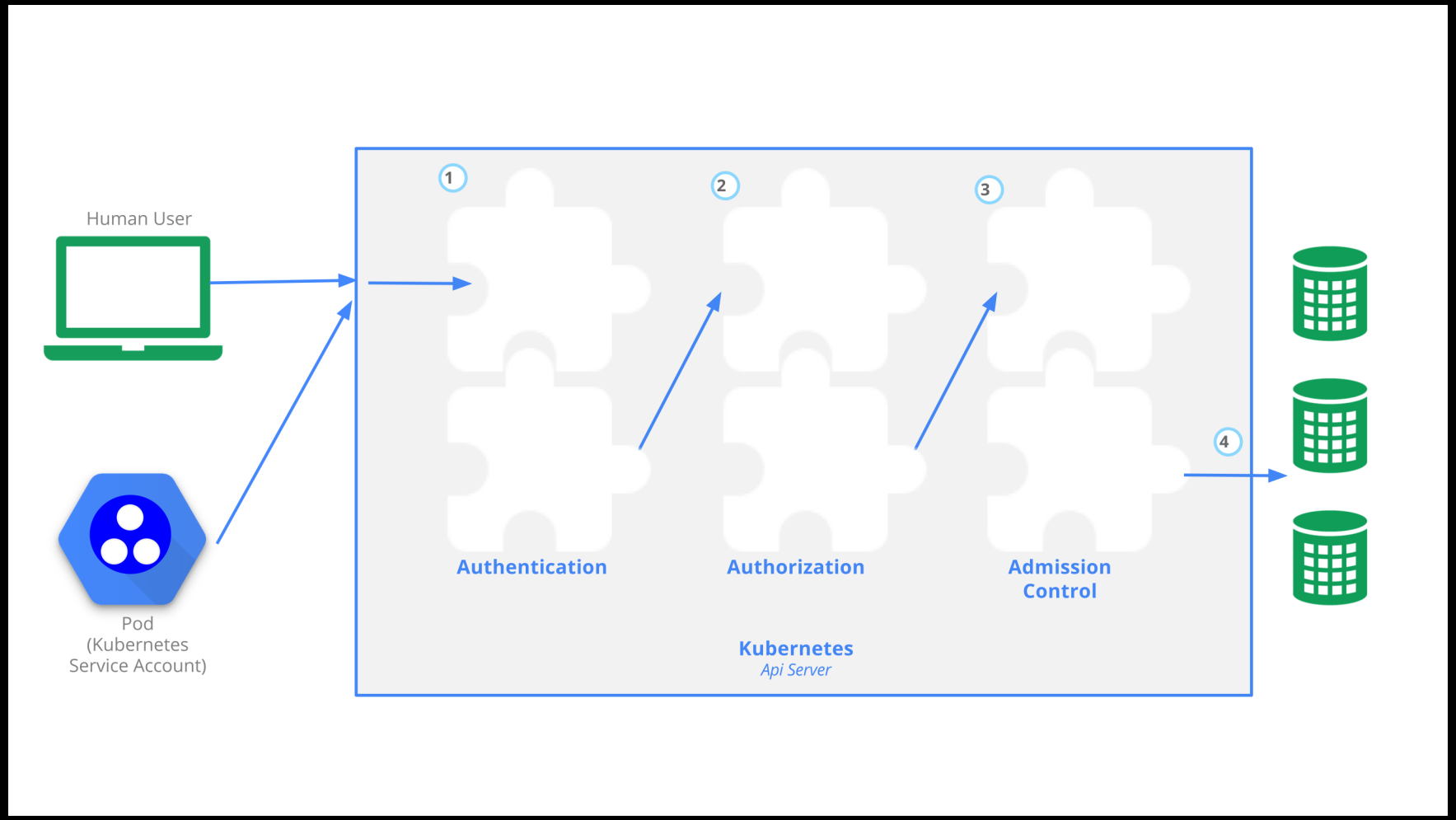
The opinions expressed herein are my own personal opinions and do not represent my employer's view in any way

Kubernetes



https://kubernetes.io/blog/2018/07/18/11-ways-not-to-get-hacked/

Access Control to Kubernetes API



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

Authentication

- Service accounts (managed by Kubernetes)
- User accounts (ideally managed by an external party)
- Multiple authentication methods (first wins)
- Strategies
  - X509 Client Certs
  - Static Token File
  - Bootstrap Tokens (Beta)
  - Static Password File
  - Service Account Tokens
  - OpenID Connect Tokens

Authorization

- Allows a subject to perform an action over an API resource
- Authorization Modes
  - Node (kubelets permissions)
  - ABAC (Attribute-based access control)
  - RBAC (Role-based access control)
  - Webhook (callback: an HTTP POST)

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

- Subjects
  - User account
  - Service account
  - Group
- Role Types
  - Role
  - ClusterRole
- Binding Types
  - RoleBinding
  - ClusterRoleBinding

Subject: User account

- Normal users are assumed to be managed by an outside, independent service
- Kubernetes does not have objects which represent normal user accounts
- Normal users cannot be added to a cluster through an API call

Demo

minikube start --extra-config=apiserver.authorization-mode=RBAC

Role and ClusterRole

In the RBAC API, a role contains rules that represent a set of permissions. Permissions are purely additive (there are no “deny” rules). A role can be defined within a namespace with a Role, or cluster-wide with a ClusterRole.

A Role can only be used to grant access to resources within a single namespace.

A ClusterRole can be used to grant the same permissions as a Role, but because they are cluster-scoped, they can also be used to grant access to:

- cluster-scoped resources (like nodes)
- non-resource endpoints (like “/healthz”, “/api”)
- namespaced resources (like pods) across all namespaces

Role and ClusterRole

```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"]
```

```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"]
```

Subresources and resource names

```
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-and-pod-logs-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: configmap-updater
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["my-configmap"]
  verbs: ["update", "get"]
```

Verbs

- get
- list
- create
- update
- patch
- watch
- proxy
- redirect
- delete
- deletecollection

apiGroups

- admissionregistration.k8s.io/v1beta1
- apiextensions.k8s.io/v1beta1
- apiregistration.k8s.io/v1
- apiregistration.k8s.io/v1beta1
- apps/v1
- apps/v1beta1
- apps/v1beta2
- v1
- …

kubectl api-versions

kubectl api-resources –o wide

RoleBinding and ClusterRoleBinding

A role binding grants the permissions defined in a role to a user or set of users. It holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted. Permissions can be granted within a namespace with a RoleBinding, or cluster-wide with a ClusterRoleBinding.

A RoleBinding may reference a Role in the same namespace, or may also reference a ClusterRole to grant the permissions to namespaced resources defined in the ClusterRole within the RoleBinding's namespace.

A ClusterRoleBinding may be used to grant permission at the cluster level and in all namespaces. You cannot modify which Role or ClusterRole a binding object refers to, the binding object must be deleted and recreated.

RoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  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
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "dave" to read secrets in the "development" namespace.
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: development # This only grants permissions within the "development" namespace.
subjects:
- kind: User
  name: dave # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to read secrets in any namespace.
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Demo

Role or ClusterRole, RoleBinding or ClusterRoleBinding

- Permission to a specific resource in a namespace
  - Role > RoleBinding
- Generic permission
  - ClusterRole > RoleBinding
- Permission across mamespaces or cluster resources
  - ClusterRole > ClusterRoleBinding

Subject: Service account

Service accounts are users managed by the Kubernetes API. They are bound to specific namespaces, and created automatically by the API server or manually through API calls. Service accounts are tied to a set of credentials stored as Secrets, which are mounted into pods allowing in-cluster processes to talk to the Kubernetes API.

- When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace
- Every namespace has a default service account resource called default
- Don’t mount service account credentials if not needed (automountServiceAccountToken: false). It can be specified in the Pod spec or ServiceAccount manifesto

Demo

Impersonation

- A user can act as another use
- Impersonation headers
  - Impersonate-User
  - Impersonate-Group
  - Impersonate-Extra-(extra name) – Requieres Impersonate-User
    - Impersonate-Extra-dn
    - Impersonate-Extra-scopes
    - …

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: impersonator
rules:
- apiGroups: [""]
  resources: ["users", "groups", "serviceaccounts"]
  verbs: ["impersonate"]
```

Demo

TL;DR

- Roles and ClusterRoles
  - apiGroup
  - resources
  - verbs
- Subjects
  - User
  - Service Account
  - Group
- RoleBinding (Role and ClusterRole) and ClusterRoleBinding (ClusterRole)
  - roleRef
  - subjects
- Pods
  - Don’t mount service account token if not needed
  - Create specific service account with the privileges needed

Questions

Always apply the least privilege principle

SAP Concur

Thank You.

Contact information:

Rafael Troncoso
rafael.troncoso@sap.com
@tuxotron

SAP Concur

Follow all of SAP Concur

Learn more at concur.com

THE BEST RUN SAP

SAP Concur

Follow all of SAP

www.sap.com/contactsap

THE BEST RUN SAP