R5 API Document

The purpose of this document is to enumerate the APIs which are exposed by Akraino Blue print project to the external projects Akraino/Non Akraino for interaction/integration. This document should be used in conjunction with the architecture document to understand APIs at a modular level and their interactions.

API1: Karmada native APIs

Karmada (Kubernetes Armada) is a Kubernetes management system that enables cloud-native applications to run across multiple Kubernetes clusters and clouds with no changes to the underlying applications. By speaking Kubernetes-native APIs and providing advanced scheduling capabilities, Karmada truly enables multi-cloud Kubernetes environment. It aims to provide turnkey automation for multi-cluster application management in multi-cloud and hybrid cloud scenarios with key features such as centralized multi-cloud management, high availability, failure recovery, and traffic scheduling.

// Cluster represents the desire state and status of a member cluster.
type Cluster struct {
metav1.TypeMeta `json:",inline"`
metav1.ObjectMeta `json:"metadata,omitempty"`

// Spec represents the specification of the desired behavior of member cluster.
Spec ClusterSpec `json:"spec"`

// Status represents the status of member cluster.
// +optional
Status ClusterStatus `json:"status,omitempty"`
}

// ClusterSpec defines the desired state of a member cluster.
type ClusterSpec struct {
// SyncMode describes how a cluster sync resources from karmada control plane.
// +kubebuilder:validation:Enum=Push;Pull
// +required
SyncMode ClusterSyncMode `json:"syncMode"`

// The API endpoint of the member cluster. This can be a hostname,
// hostname:port, IP or IP:port.
// +optional
APIEndpoint string `json:"apiEndpoint,omitempty"`

// SecretRef represents the secret contains mandatory credentials to access the member cluster.
// The secret should hold credentials as follows:
// - secret.data.token
// - secret.data.caBundle
// +optional
SecretRef *LocalSecretReference `json:"secretRef,omitempty"`

// InsecureSkipTLSVerification indicates that the karmada control plane should not confirm the validity of the serving
// certificate of the cluster it is connecting to. This will make the HTTPS connection between the karmada control
// plane and the member cluster insecure.
// Defaults to false.
// +optional
InsecureSkipTLSVerification bool `json:"insecureSkipTLSVerification,omitempty"`

// ProxyURL is the proxy URL for the cluster.
// If not empty, the karmada control plane will use this proxy to talk to the cluster.
// More details please refer to: https://github.com/kubernetes/client-go/issues/351
// +optional
ProxyURL string `json:"proxyURL,omitempty"`

// Provider represents the cloud provider name of the member cluster.
// +optional
Provider string `json:"provider,omitempty"`

// Region represents the region of the member cluster locate in.
// +optional
Region string `json:"region,omitempty"`

// Zone represents the zone of the member cluster locate in.
// +optional
Zone string `json:"zone,omitempty"`

```
// Taints attached to the member cluster.
// Taints on the cluster have the "effect" on
// any resource that does not tolerate the Taint.
// +optional
Taints []corev1.Taint `json:"taints,omitempty"`
}

// ClusterStatus contains information about the current status of a
// cluster updated periodically by cluster controller.
type ClusterStatus struct {
// KubernetesVersion represents version of the member cluster.
// +optional
KubernetesVersion string `json:"kubernetesVersion,omitempty"`

// APIEnablements represents the list of APIs installed in the member cluster.
// +optional
APIEnablements []APIEnablement `json:"apiEnablements,omitempty"`

// Conditions is an array of current cluster conditions.
// +optional
Conditions []metav1.Condition `json:"conditions,omitempty"`

// NodeSummary represents the summary of nodes status in the member cluster.
// +optional
NodeSummary *NodeSummary `json:"nodeSummary,omitempty"`

// ResourceSummary represents the summary of resources in the member cluster.
// +optional
ResourceSummary *ResourceSummary `json:"resourceSummary,omitempty"`
}

// ResourceSummary represents the summary of resources in the member cluster.
type ResourceSummary struct {
// Allocatable represents the resources of a cluster that are available for scheduling.
// Total amount of allocatable resources on all nodes.
// +optional
Allocatable corev1.ResourceList `json:"allocatable,omitempty"`
// Allocating represents the resources of a cluster that are pending for scheduling.
// Total amount of required resources of all Pods that are waiting for scheduling.
// +optional
Allocating corev1.ResourceList `json:"allocating,omitempty"`
// Allocated represents the resources of a cluster that have been scheduled.
// Total amount of required resources of all Pods that have been scheduled to nodes.
// +optional
Allocated corev1.ResourceList `json:"allocated,omitempty"`
}

// ClusterList contains a list of member cluster
type ClusterList struct {
metav1.TypeMeta `json:",inline"`
metav1.ListMeta `json:"metadata,omitempty"`

// Items holds a list of Cluster.
Items []Cluster `json:"items"`
}

// PropagationPolicy represents the policy that propagates a group of resources to one or more clusters.
type PropagationPolicy struct {
metav1.TypeMeta `json:",inline"`
metav1.ObjectMeta `json:"metadata,omitempty"`

// Spec represents the desired behavior of PropagationPolicy.
// +required
Spec PropagationSpec `json:"spec"`
}

// PropagationSpec represents the desired behavior of PropagationPolicy.
type PropagationSpec struct {
// ResourceSelectors used to select resources.
// +required
ResourceSelectors []ResourceSelector `json:"resourceSelectors"`

// Association tells if relevant resources should be selected automatically.
// e.g. a ConfigMap referred by a Deployment.
// default false.
// +optional
Association bool `json:"association,omitempty"`
```

```
// Placement represents the rule for select clusters to propagate resources.
// +optional
Placement Placement `json:"placement,omitempty"`

// DependentOverrides represents the list of overrides(OverridePolicy)
// which must present before the current PropagationPolicy takes effect.
//
// It used to explicitly specify overrides which current PropagationPolicy rely on.
// A typical scenario is the users create OverridePolicy(ies) and resources at the same time,
// they want to ensure the new-created policies would be adopted.
//
// Note: For the overrides, OverridePolicy(ies) in current namespace and ClusterOverridePolicy(ies),
// which not present in this list will still be applied if they matches the resources.
// +optional
DependentOverrides []string `json:"dependentOverrides,omitempty"`

// SchedulerName represents which scheduler to proceed the scheduling.
// If specified, the policy will be dispatched by specified scheduler.
// If not specified, the policy will be dispatched by default scheduler.
// +optional
SchedulerName string `json:"schedulerName,omitempty"`
}
```

// ResourceSelector the resources will be selected.

```
type ResourceSelector struct {
// APIVersion represents the API version of the target resources.
// +required
APIVersion string `json:"apiVersion"`

// Kind represents the Kind of the target resources.
// +required
Kind string `json:"kind"`

// Namespace of the target resource.
// Default is empty, which means inherit from the parent object scope.
// +optional
Namespace string `json:"namespace,omitempty"`

// Name of the target resource.
// Default is empty, which means selecting all resources.
// +optional
Name string `json:"name,omitempty"`

// A label query over a set of resources.
// If name is not empty, labelSelector will be ignored.
// +optional
LabelSelector *metav1.LabelSelector `json:"labelSelector,omitempty"`
}
```

// FieldSelector is a field filter.

```
type FieldSelector struct {
// A list of field selector requirements.
MatchExpressions []corev1.NodeSelectorRequirement `json:"matchExpressions,omitempty"`
}
```

// Placement represents the rule for select clusters.

```
type Placement struct {
// ClusterAffinity represents scheduling restrictions to a certain set of clusters.
// If not set, any cluster can be scheduling candidate.
// +optional
ClusterAffinity *ClusterAffinity `json:"clusterAffinity,omitempty"`

// ClusterTolerations represents the tolerations.
// +optional
ClusterTolerations []corev1.Toleration `json:"clusterTolerations,omitempty"`

// SpreadConstraints represents a list of the scheduling constraints.
// +optional
SpreadConstraints []SpreadConstraint `json:"spreadConstraints,omitempty"`

// ReplicaScheduling represents the scheduling policy on dealing with the number of replicas
// when propagating resources that have replicas in spec (e.g. deployments, statefulsets) to member clusters.
// +optional
ReplicaScheduling *ReplicaSchedulingStrategy `json:"replicaScheduling,omitempty"`
}
```

// SpreadFieldValue is the type to define valid values for SpreadConstraint.SpreadByField

```
type SpreadFieldValue string
```

```
// Available fields for spreading are: cluster, region, zone, and provider.
const (
SpreadByFieldCluster SpreadFieldValue = "cluster"
SpreadByFieldRegion SpreadFieldValue = "region"
SpreadByFieldZone SpreadFieldValue = "zone"
SpreadByFieldProvider SpreadFieldValue = "provider"
)

// SpreadConstraint represents the spread constraints on resources.
type SpreadConstraint struct {
// SpreadByField represents the fields on Karmada cluster API used for
// dynamically grouping member clusters into different groups.
// Resources will be spread among different cluster groups.
// Available fields for spreading are: cluster, region, zone, and provider.
// SpreadByField should not co-exist with SpreadByLabel.
// If both SpreadByField and SpreadByLabel are empty, SpreadByField will be set to "cluster" by system.
// +kubebuilder:validation:Enum=cluster;region;zone;provider
// +optional
SpreadByField SpreadFieldValue `json:"spreadByField,omitempty"`

// SpreadByLabel represents the label key used for
// grouping member clusters into different groups.
// Resources will be spread among different cluster groups.
// SpreadByLabel should not co-exist with SpreadByField.
// +optional
SpreadByLabel string `json:"spreadByLabel,omitempty"`

// MaxGroups restricts the maximum number of cluster groups to be selected.
// +optional
MaxGroups int `json:"maxGroups,omitempty"`

// MinGroups restricts the minimum number of cluster groups to be selected.
// Defaults to 1.
// +optional
MinGroups int `json:"minGroups,omitempty"`
}

// ClusterAffinity represents the filter to select clusters.
type ClusterAffinity struct {
// LabelSelector is a filter to select member clusters by labels.
// If non-nil and non-empty, only the clusters match this filter will be selected.
// +optional
LabelSelector *metav1.LabelSelector `json:"labelSelector,omitempty"`

// FieldSelector is a filter to select member clusters by fields.
// If non-nil and non-empty, only the clusters match this filter will be selected.
// +optional
FieldSelector *FieldSelector `json:"fieldSelector,omitempty"`

// ClusterNames is the list of clusters to be selected.
// +optional
ClusterNames []string `json:"clusterNames,omitempty"`

// ExcludedClusters is the list of clusters to be ignored.
// +optional
ExcludeClusters []string `json:"exclude,omitempty"`
}

// ReplicaSchedulingType describes scheduling methods for the "replicas" in a resouce.
type ReplicaSchedulingType string

const (
// ReplicaSchedulingTypeDuplicated means when propagating a resource,
// each candidate member cluster will directly apply the original replicas.
ReplicaSchedulingTypeDuplicated ReplicaSchedulingType = "Duplicated"
// ReplicaSchedulingTypeDivided means when propagating a resource,
// each candidate member cluster will get only a part of original replicas.
ReplicaSchedulingTypeDivided ReplicaSchedulingType = "Divided"
)

// ReplicaDivisionPreference describes options of how replicas can be scheduled.
type ReplicaDivisionPreference string
```

```
const (
// ReplicaDivisionPreferenceAggregated divides replicas into clusters as few as possible,
// while respecting clusters' resource availabilities during the division.
ReplicaDivisionPreferenceAggregated ReplicaDivisionPreference = "Aggregated"
// ReplicaDivisionPreferenceWeighted divides replicas by weight according to WeightPreference.
ReplicaDivisionPreferenceWeighted ReplicaDivisionPreference = "Weighted"
)

// ReplicaSchedulingStrategy represents the assignment strategy of replicas.
type ReplicaSchedulingStrategy struct {
// ReplicaSchedulingType determines how the replicas is scheduled when karmada propagating
// a resource. Valid options are Duplicated and Divided.
// "Duplicated" duplicates the same replicas to each candidate member cluster from resource.
// "Divided" divides replicas into parts according to number of valid candidate member
// clusters, and exact replicas for each cluster are determined by ReplicaDivisionPreference.
// +kubebuilder:validation:Enum=Duplicated;Divided
// +optional
ReplicaSchedulingType ReplicaSchedulingType `json:"replicaSchedulingType,omitempty"`

// ReplicaDivisionPreference determines how the replicas is divided
// when ReplicaSchedulingType is "Divided". Valid options are Aggregated and Weighted.
// "Aggregated" divides replicas into clusters as few as possible,
// while respecting clusters' resource availabilities during the division.
// "Weighted" divides replicas by weight according to WeightPreference.
// +kubebuilder:validation:Enum=Aggregated;Weighted
// +optional
ReplicaDivisionPreference ReplicaDivisionPreference `json:"replicaDivisionPreference,omitempty"`

// WeightPreference describes weight for each cluster or for each group of cluster
// If ReplicaDivisionPreference is set to "Weighted", and WeightPreference is not set, scheduler will weight all clusters the same.
// +optional
WeightPreference *ClusterPreferences `json:"weightPreference,omitempty"`
}

// PropagationPolicyList contains a list of PropagationPolicy.
type PropagationPolicyList struct {
metav1.TypeMeta `json:",inline"`
metav1.ListMeta `json:"metadata,omitempty"`
Items []PropagationPolicy `json:"items"`
}

// ClusterPropagationPolicy represents the cluster-wide policy that propagates a group of resources to one or more clusters.
// Different with PropagationPolicy that could only propagate resources in its own namespace, ClusterPropagationPolicy
// is able to propagate cluster level resources and resources in any namespace other than system reserved ones.
// System reserved namespaces are: karmada-system, karmada-cluster, karmada-es-*.
type ClusterPropagationPolicy struct {
metav1.TypeMeta `json:",inline"`
metav1.ObjectMeta `json:"metadata,omitempty"`

// Spec represents the desired behavior of ClusterPropagationPolicy.
// +required
Spec PropagationSpec `json:"spec"`
}

// ClusterPropagationPolicyList contains a list of ClusterPropagationPolicy.
type ClusterPropagationPolicyList struct {
metav1.TypeMeta `json:",inline"`
metav1.ListMeta `json:"metadata,omitempty"`
Items []ClusterPropagationPolicy `json:"items"`
}
```