Binary Provisioning Agent (BPA) Operator Specs

Goals

This wiki describes the specifications for designing the Binary Provisioning Agent required for the Integrated Cloud Native Akraino project.

Overview of BPA

The BPA is part of the infra local controller which runs as a bootstrap k8s cluster in the ICN project. As described in Integrated Cloud Native Akraino project , the purpose of the BPA is to install packages that cannot be installed using kubectl. It will be called once the operating system (Linux) has been installed in the compute nodes by the baremetal operator. The Binary Provisioning Agent will carry out the following functions;

1. Create the hosts.ini file required to install kubernetes on compute nodes in order to create a cluster using kubespray. It uses the roles specified in the provisioning custom resource
2. Instantiate the binary package installation and get the status of the installation
3. Install the packages on newly added compute nodes
4. Update package versions in compute nodes that require the update
5. Store private keys

For more information on the BPA functions, check out the ICN Akraino project link above

Implementation

We do not intend to make any changes to the existing kubernetes API in order to implement the specifications described in this document. We will simply be extending the Kubernetes API using Custom Resource Definition as described here and then creating a custom controller that will handle the requirements of our provisioning Agent custom resource.

Overview of Proposed Workflow for provisioning CRD

Prerequisites: This workflow assumes that the baremetal CR and baremetal operator have been created and has successfully installed the compute nodes with Linux OS. It also assumes that the BPA controller is running.

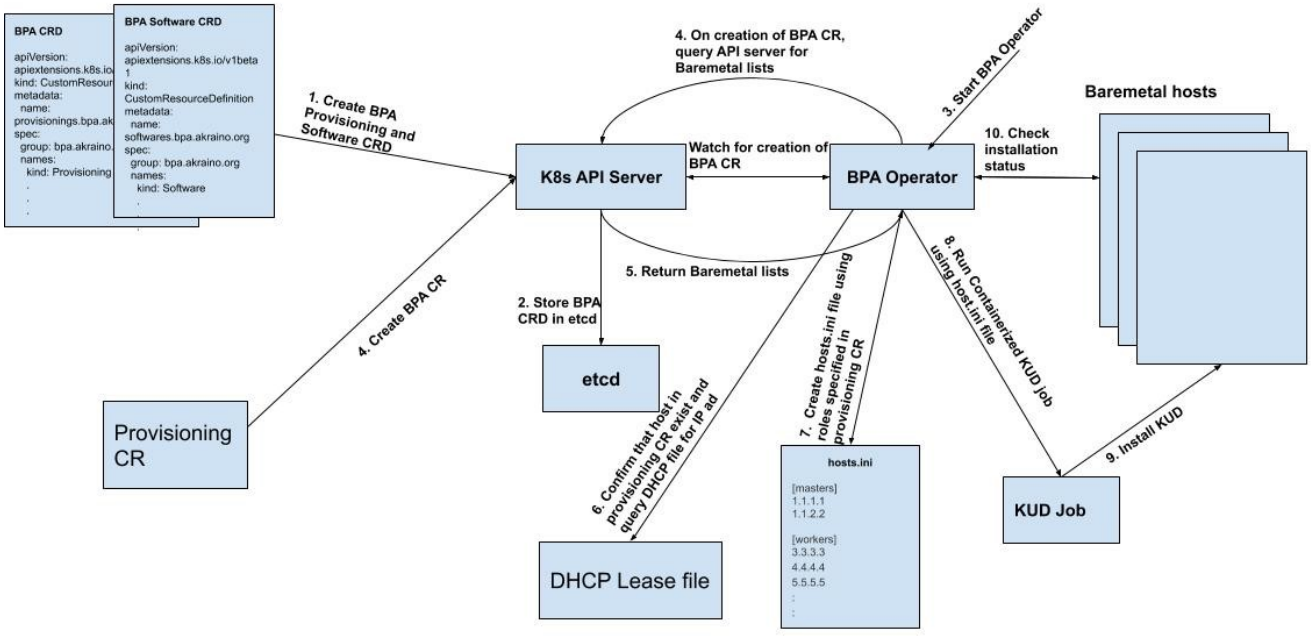

Fig 1: Illustration of the proposed workflow

Workflow Summary/Description

1. Create Provisioning CRD and Software CRD

2. The CRDs are stored in ETCD
3. Start the BPA controller (It then watches for the creation of either a software CR or provisioning CR (We will be focusing on the provisioning CR) here).
4. Create an instance of the Provisioning Custom Resource, this can be done at any instance once the BPA operator is running
5. The BPA Operator continues to watch the k8s API server and once it sees that a new BPA CR object has been created, it queries the k8s API server for the Baremetal hosts lists. The baremetal hosts lists contains information about the compute nodes provisioned.
6. Confirm that all the hosts specified in the provisioning CR exist in the Baremetal hosts list, then query the DHCP lease file using the MAC address of each host to get the corresponding IP addresses.
7. Create the hosts.ini file using the roles specified in the provisioning CR and the MAC addresses from 6 above.
8. Once the hosts.ini file is created, start the KUD job.
9. KUD job installs kud in the host.
10. BPA operator spawns a thread that continues to check the status of the KUD job.
11. Once the KUD installation is completed, the BPA operator, creates a configmap for that cluster, the configmap contains a mapping of the IP address to the host label specified in the provisioning CR. (Step 11 is not shown in the diagram). This configmap will be used when the BPA operator is installing software specified in the software CR (see BPA Software CR Specs).

BPA CRD

The BPA CRD tells the Kubernetes API how to expose the provisioning custom resource object. The CRD yaml file is applied using

"kubectl create -f bpa_v1alpha1_provisioning_crd.yaml" See below for the CRD definition.

BPA CRD Yaml File (*_crd.yaml)

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: provisionings.bpa.akraino.org
spec:
  group: bpa.akraino.org
  names:
    kind: Provisioning
    listKind: ProvisioningList
    plural: provisionings
    singular: provisioning
    shortNames:
    - bpa
  scope: Namespaced
  subresources:
    status: {}
  validation:
    openAPIV3Schema:
      properties:
        apiVersion:
          description:
          type: string
        kind:
          description:
          type: string
        metadata:
          type: object
        spec:
          type: object
        status:
          type: object
  version: v1alpha1
  versions:
  - name: v1alpha1
    served: true
    storage: true
```

Provisioning Agent Object Definition(*_types,go)

The provisioning_types.go file is the API for the provisioning agent custom resource.

```
// ProvisioningSpec defines the desired state of Provisioning
// +k8s:openapi-gen=true
type ProvisioningSpec struct {
        // INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
        // Important: Run "operator-sdk generate k8s" to regenerate code after modifying this file
        // Add custom validation using kubebuilder tags: https://book-v1.book.kubebuilder.io/beyond_basics
/generating_crd.html
        Masters []map[string]Master  `json:"masters,omitempty"`
        Workers []map[string]Worker  `json:"workers,omitempty"`
        KUDPlugins []string `json:"KUDPlugins,omitempty"`
}

// ProvisioningStatus defines the observed state of Provisioning
// +k8s:openapi-gen=true
type ProvisioningStatus struct {
        // INSERT ADDITIONAL STATUS FIELD - define observed state of cluster
        // Important: Run "operator-sdk generate k8s" to regenerate code after modifying this file
        // Add custom validation using kubebuilder tags: https://book-v1.book.kubebuilder.io/beyond_basics
/generating_crd.html
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// Provisioning is the Schema for the provisionings API
// +k8s:openapi-gen=true
// +kubebuilder:subresource:status
type Provisioning struct {
        metav1.TypeMeta   `json:",inline"`
        metav1.ObjectMeta `json:"metadata,omitempty"`

        Spec   ProvisioningSpec   `json:"spec,omitempty"`
        Status ProvisioningStatus `json:"status,omitempty"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

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

// master struct contains resource requirements for a master node
type Master struct {
        MACaddress string `json:"mac-address,omitempty"`
        CPU int32  `json:"cpu,omitempty"`
        Memory string  `json:"memory,omitempty"`
}

// worker struct contains resource requirements for a worker node
type Worker struct {
        MACaddress string `json:"mac-address,omitempty"`
        CPU int32 `json:"cpu,omitempty"`
        Memory string  `json:"memory,omitempty"`
        SRIOV bool  `json:"sriov,omitempty"`
        QAT  bool      `json:"qat,omitempty"`
}

func init() {
        SchemeBuilder.Register(&Provisioning{}, &ProvisioningList{})
}
```

The variables in the ProvisioningSpec struct are used to create the data structures in the yaml spec for the custom resource. Three variables are added to the ProvisioningSpec struct;

1. **Masters**: This variable will contain an array of Master objects. The master struct as defined in the *-types.go file above contains CPU and memory information, this information would be used by the BPA operator to determine which compute nodes to assign the role of Master to when it gets the baremetal list from the API server.
2. **Workers**: This variable will contain an array of Worker objects. Similar to the case of the Masters variables, the Worker struct will contain resource requirements for the Worker nodes and the BPA operator will use this information to determine which hosts to assign the role of worker.
3. **KUDPlugins:** This variable will contain the list of KUD plugins to be installed with KUD in the cluster

Sample Provisioning CR YAML files

```
apiVersion: bpa.akraino.org/v1alpha1
kind: Provisioning
metadata:
  name: provisioning-sample
  labels:
    cluster: cluster-abc
    owner: c1
spec:
  masters:
   - master:
      mac-address: 00:c6:14:04:61:b2
  workers:
    - worker-1:
       mac-address: 00:c6:14:04:61:b2
    - worker-2:
       mac-address: 00:c2:12:03:62:b1
```

```
apiVersion: bpa.akraino.org/v1alpha1
kind: Provisioning
metadata:
  name: sample-kud-plugins
  labels:
    cluster: cluster-efg
    owner: c2
spec:
  masters:
    - master-1:
        mac-address: 00:e1:ba:ce:df:bd
  KUDPlugins:
    - onap4k8s
```

```
apiVersion: bpa.akraino.org/v1alpha1
kind: Provisioning
metadata:
  name: provisioning-sample
  labels:
    cluster: cluster-xyz
    owner: c2
spec:
  masters:
   - master-1:
      cpu: 10
      memory: 4Gi
      mac-address: 00:c5:16:05:61:b2
   - master-2:
      cpu: 10
      memory: 4Gi
      mac-address: 00:c2:14:06:61:b5
  workers:
   - worker:
      cpu: 20
      memory: 8Gi
      mac-address: 00:c6:14:04:61:b2
```

The YAML file above can be used to create a provisioning custom resource which is an instance of the provisioning CRD describes above. The spec. master field corresponds to the Masters variable in the ProvisioningSpec struct of the *-types.go file, while the spec.worker field corresponds to the Workers variable in the ProvisioningSpec struct of the *-types.go file.

Currently the cpu and memory fields are not used by the BPA operator code. More Provisioning CRs can be found in here.

Sample Baremetal Lists from Query

```
apiVersion: v1
items:
- apiVersion: metal3.io/v1alpha1
  kind: BareMetalHost
  metadata:
    creationTimestamp: "2019-07-20T01:43:19Z"
    finalizers:
    - baremetalhost.metal3.io
    generation: 2
    name: demo-provisioning
    namespace: metal3
    resourceVersion: "35002"
    selfLink: /apis/metal3.io/v1alpha1/namespaces/metal3/baremetalhosts/demo-provisioning
    uid: 3b22014e-9252-4f15-89a5-67f96e1a07a2
  spec:
    bmc:
      address: ipmi://172.31.1.17
      credentialsName: demo-provisioning-bmc-secret
    description: ""
    externallyProvisioned: false
    hardwareProfile: ""
    image:
      checksum: http://172.22.0.1/images/bionic-server-cloudimg-amd64.md5sum
      url: http://172.22.0.1/images/bionic-server-cloudimg-amd64.qcow2
    online: true
  status:
    errorMessage: ""
    goodCredentials:
      credentials:
        name: demo-provisioning-bmc-secret
        namespace: metal3
      credentialsVersion: "30393"
    hardware:
      cpu:
        arch: x86_64
        clockMegahertz: 3700
        count: 72
        flags:
        -  ….
        - xtopology
        - xtpr
        model: Intel(R) Xeon(R) Gold 6140M CPU @ 2.30GHz
      firmware:
        bios:
          date: 11/07/2018
          vendor: Intel Corporation
          version: SE5C620.86B.00.01.0015.110720180833
      hostname: localhost.localdomain
      nics:
      - ip: ""
        mac: 3c:fd:fe:9c:88:60
        model: 0x8086 0x1572
        name: eth0
        pxe: false
        speedGbps: 0
        vlanId: 0
      - ip: 172.22.0.55
        mac: a4:bf:01:64:86:6f
        model: 0x8086 0x37d2
        name: eth5
        pxe: true
        speedGbps: 0
        vlanId: 0
        …
      ramMebibytes: 262144
      storage:
```

```
      - hctl: "6:0:0:0"
        model: INTEL SSDSC2KB48
        name: /dev/sda
        rotational: false
        serialNumber: BTYF8290022M480BGN
        sizeBytes: 480103981056
        vendor: ATA
        wwn: "0x55cd2e414fc888c1"
        wwnWithExtension: "0x55cd2e414fc888c1"
      - hctl: "7:0:0:0"
        model: INTEL SSDSC2KB48
        name: /dev/sdb
        rotational: false
        serialNumber: BTYF83160FDB480BGN
        sizeBytes: 480103981056
        vendor: ATA
        wwn: "0x55cd2e414fd7b5a3"
        wwnWithExtension: "0x55cd2e414fd7b5a3"
      systemVendor:
        manufacturer: Intel Corporation
        productName: S2600WFT (SKU Number)
        serialNumber: BQPW84200264
    hardwareProfile: unknown
    lastUpdated: "2019-07-20T02:41:30Z"
    operationalStatus: OK
    poweredOn: false
    provisioning:
      ID: 94fa2511-3cb1-4372-ab42-9c377db8aeca
      image:
        checksum: ""
        url: ""
      state: provisioning
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

In addition, we would also have two other CRDs that the BPA would use to perform its functions;

1. Software CRD
2. Cluster CRD

Software CRD

The software CRD will install the required software, drivers and perform software updates. See BPA Software CR Specs.

Draft Software CRD

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: software.bpa.akraino.org
spec:
  group: bpa.akraino.org
  names:
    kind: software
    listKind: softwarerList
    plural: software
    singular: software
    shortNames:
    - su
  scope: Namespaced
  subresources:
    status: {}
```

```
  validation:
    openAPIV3Schema:
      properties:
        apiVersion:
          description:
          type: string
        kind:
          description:
          type: string
        metadata:
          type: object
        spec:
          type: object
        status:
          type: object
  version: v1alpha1
  versions:
  - name: v1alpha1
    served: true
    storage: true
```

Sample Software CR YAML files

```
apiVersion: bpa.akraino.org/v1alpha1
kind: Software
metadata:
  labels:
    cluster: cluster-xyz
    owner: c1
  name: example-software
spec:
  masterSoftware:
    - curl
    - htop
    - jq:
        version: 1.5+dfsg-1ubuntu0.1
    - maven:
        version: 3.3.9-3
  workerSoftware:
    - curl
    - htop
    - tmux
    - jq
```

Cluster CRD

The cluster CRD will have the Cluster name and contain the provisioning CR and/or the software CR for the specified cluster

Draft Cluster CRD

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: cluster.bpa.akraino.org
spec:
  group: bpa.akraino.org
  names:
    kind: cluster
    listKind: clusterList
    plural: clusters
    singular: cluster
    shortNames:
    - cl
  scope: Namespaced
```

```
  subresources:
    status: {}
  validation:
    openAPIV3Schema:
      properties:
        apiVersion:
          description:
          type: string
        kind:
          description:
          type: string
        metadata:
          type: object
        spec:
          type: object
        status:
          type: object
  version: v1alpha1
  versions:
  - name: v1alpha1
    served: true
    storage: true
```

Sample Cluster CR YAML files

```
apiVersion: bpa.akraino.org/v1alpha1
kind: cluster
metadata:
  name: cluster-sample
  labels:
    cluster: cluster-abc
    owner: c1
spec:
   provisioningCR: "provisioning-sample"
   softwareCR: "software-sample"
```

Future Work

This proposal would make it possible to assign roles to nodes based on the features discovered. The baremetal operator list returns much more information about the nodes, we would be able to extend the feature to allow the operator determine the right nodes to use complex requirements such as CPU model, memory, CPU..etc This would feed into Hardware Platform Awareness (HPA)

References

1. https://wiki.akraino.org/pages/viewpage.action?pageId=11995877&show-miniview
2. https://kubernetes.io/docs/tasks/access-kubernetes-api/custom-resources/custom-resource-definitions/#advanced-topics

Presentation:


Akraino-Intergra...estedK8s HA.pptx