# Baremetal Operator

# Table of Contents



# 1. API and Resource Definitions

## 1.1. BareMetalHost

Metal³ introduces the concept of BareMetalHost resource, which defines a physical host and its properties. The BareMetalHost embeds two well differentiated sections, the bare metal host specification and its current status.

### 1.1.1. BareMetalHost spec

The *BareMetalHost's spec* defines the desire state of the host. It contains mainly, but not only, provisioning details.

**bmc**

The `bmc` fields contain the connection information for the BMC (Baseboard Management Controller) on the host.

The sub-fields are

- *address* — The URL for communicating with the BMC controller, based on the provider being used. See below for more details.
- *credentialsName* — A reference to a *secret* containing the username and password for the BMC.
- *disableCertificateVerification* — A boolean to skip certificate validation when true.

BMC URLs vary based on the type of BMC and the protocol used to communicate with them.

- IPMI
  - `ipmi://<host>:<port>`, an unadorned `<host>:<port>` is also accepted and the port is optional, if using the default one (623).
- Dell iDRAC
  - `idrac://` (or `idrac+http://` to disable TLS).
  - `idrac-virtualmedia://` to use virtual media instead of PXE for attaching the provisioning image to the host.
- Fujitsu iRMC
  - `irmc://<host>:<port>`, where `<port>` is optional if using the default.
- HUAWEI ibmc
  - `ibmc://<host>:<port>` (or `ibmc+http://<host>:<port>` to disable TLS)
- HPE iLO 4
  - `ilo4://<host>:<port>` for iLO 4 based systems and the port is optional, if using the default one (443).
- HPE iLO 5
  - `ilo5://<host>:<port>` for iLO 5 based systems and the port is optional, if using the default one (443).
- iLO 5 Redfish
  - `ilo5-redfish://` (or `ilo5-redfish+http://` to disable TLS), the hostname or IP address, and the path to the system ID are required, for example `ilo5-redfish://myhost.example/redfish/v1/Systems/MySystemExample`
- Redfish
  - `redfish://` (or `redfish+http://` to disable TLS)
  - `redfish-virtualmedia://` to use virtual media instead of PXE for attaching the provisioning image to the host.
  - The hostname or IP address, and the path to the system ID are required for all variants. For example `redfish://myhost.example/redfish/v1/Systems/System.Embedded.1` or `redfish://myhost.example/redfish/v1/Systems/1`

#### online

A boolean indicating whether the host should be powered on (true) or off (false). Changing this value will trigger a change in power state on the physical host.

#### consumerRef

A reference to another resource that is using the host, it could be empty if the host is not being currently used. For example, a *Machine* resource when the host is being used by the *machine-api*.

#### externallyProvisioned

A boolean indicating whether the host provisioning and deprovisioning are managed externally. When set, the host's power status and hardware inventory will be monitored but no provisioning or deprovisioning operations are performed on the host.

#### image

Holds details for the image to be deployed on a given host.

The sub-fields are

- *url* — The URL of an image to deploy to the host.
- *checksum* — The actual checksum or a URL to a file containing the checksum for the image at *image.url*.
- *checksumType* — Checksum algorithms can be specified. Currently only `md5`, `sha256`, `sha512` are recognized. If nothing is specified `md5` is assumed.
- *format* — This is the disk format of the image. It can be one of `raw`, `qcow2`, `vdi`, `vmdk`, or be left unset. Setting it to raw enables raw image streaming in Ironic agent for that image.

Even though the image sub-fields are required by Ironic, when the host provisioning is managed externally via `externallyProvisioned: true`, and power control isn't needed, the fields can be left empty.

#### userData

A reference to the Secret containing the cloudinit user data and its namespace, so it can be attached to the host before it boots for configuring different aspects of the OS (like networking, storage, ...).

#### networkData

A reference to the Secret containing the network configuration data (e.g. network_data.json) and its namespace, so it can be attached to the host before it boots to set network up

#### description

A human-provided string to help identify the host.

**hardwareProfile**

**This field is deprecated. See rootDeviceHints instead.**

The name of the hardware profile to use. The following are the current supported `hardwareProfile` settings and their corresponding root devices.

| hardwareProfile | Root Device |
|---|---|
| `unknown` | /dev/sda |
| `libvirt` | /dev/vda |
| `dell` | HCTL: 0:0:0:0 |
| `dell-raid` | HCTL: 0:2:0:0 |
| `openstack` | /dev/vdb |

**NOTE:** These are subject to change.

**rootDeviceHints**

Guidance for how to choose the device to receive the image being provisioned. The storage devices are examined in the order they are discovered during inspection and the hint values are compared to the inspected values. The first discovered device that matches is used. Hints can be combined, and if multiple hints are provided then a device must match all hints in order to be selected.

The sub-fields are

- *deviceName* — A string containing a Linux device name like `/dev/vda`. The hint must match the actual value exactly.
- *hctl* — A string containing a SCSI bus address like `0:0:0:0`. The hint must match the actual value exactly.
- *model* — A string containing a vendor-specific device identifier. The hint can be a substring of the actual value.
- *vendor* — A string containing the name of the vendor or manufacturer of the device. The hint can be a substring of the actual value.
- *serialNumber* — A string contianing the device serial number. The hint must match the actual value exactly.
- *minSizeGigabytes* — An integer representing the minimum size of the device in Gigabytes.
- *wwn* — A string containing the unique storage identifier. The hint must match the actual value exactly.
- *wwnWithExtension* — A string containing the unique storage identifier with the vendor extension appended. The hint must match the actual value exactly.
- *wwnVendorExtension* — A string containing the unique vendor storage indentifier. The hint must match the actual value exactly.
- *rotational* — A boolean indicating whether the device should be a rotating disk (`true`) or not (`false`).

### 1.1.2. BareMetalHost status

Moving onto the next block, the *BareMetalHost's status* which represents the host's current state. Including tested credentials, current hardware details, etc.

#### goodCredentials

A reference to the secret and its namespace holding the last set of BMC credentials the system was able to validate as working.

#### triedCredentials

A reference to the secret and its namespace holding the last set of BMC credentials that were sent to the provisioning backend.

#### lastUpdated

The timestamp of the last time the status of the host was updated.

#### operationalStatus

The status of the server. Value is one of the following:

- *OK* — Indicates all the details for the host are known and working, meaning the host is correctly configured and manageable.
- *discovered* — Implies some of the host's details are either not working correctly or missing. For example, the BMC address is known but the login credentials are not.
- *error* — Indicates the system found some sort of irrecuperable error. Refer to the *errorMessage* field in the status section for more details.

#### errorMessage

Details of the last error reported by the provisioning backend, if any.

#### hardware

The details for hardware capabilities discovered on the host. These are filled in by the provisioning agent when the host is registered.

The sub-fields are

- *nics* — List of network interfaces for the host.
  - *name* — A string identifying the network device, e.g. *nic-1*.
  - *mac* — The MAC address of the NIC.
  - *ip* — The IP address of the NIC, if one was assigned when the discovery agent ran.
  - *speedGbps* — The speed of the device in Gbps.
  - *vlans* — A list holding all the VLANs available for this NIC.
  - *vlanId* — The untagged VLAN ID.

    - *pxe* — Whether the NIC is able to boot using PXE.
- *storage* — List of storage (disk, SSD, etc.) available to the host.
    - *name* — A string identifying the storage device, e.g. *disk 1 (boot)*.
    - *rotational* — Either true or false, indicates whether the disk is rotational.
    - *sizeBytes* — Size of the storage device.
    - *serialNumber* — The device's serial number.
- *cpu* — Details of the CPU(s) in the system.
    - *arch* — The architecture of the CPU.
    - *model* — The model string.
    - *clockMegahertz* — The speed in GHz of the CPU.
    - *flags* — List of CPU flags, e.g. 'mmx','sse','sse2','vmx', ...
    - *count* — Amount of these CPUs available in the system.
- *firmware* — Contains BIOS information like for instance its *vendor* and *version*.
- *systemVendor* — Contains information about the host's *manufacturer*, the *productName* and *serialNumber*.
- *ramMebibytes* — The host's amount of memory in Mebibytes.

**hardwareProfile (status)**

**This field is deprecated. See rootDeviceHints instead.**

The name of the hardware profile that matches the hardware discovered on the host based on the details saved to the *Hardware* section. If the hardware does not match any known profile, the value `unknown` will be set on this field and is used by default. In practice, this only affects which device the OS image will be written to. The following are the current supported `hardwareProfile` settings and their corresponding root devices.

| hardwareProfile | Root Device |
|---|---|
| `unknown` | /dev/sda |
| `libvirt` | /dev/vda |
| `dell` | HCTL: 0:0:0:0 |
| `dell-raid` | HCTL: 0:2:0:0 |
| `openstack` | /dev/vdb |

**NOTE:** These are subject to change.

**poweredOn**

Boolean indicating whether the host is powered on.

See *online* on the *BareMetalHost's Spec*.

**provisioning**

Settings related to deploying an image to the host.

- *state* — The current state of any ongoing provisioning operation. The following are the currently supported ones:
  - <*empty string*> — There is no provisioning happening, at the moment.
  - *registration error* — The details for the host's BMC are either incorrect or incomplete therfore the host could not be managed.
  - *registering* — The host's BMC details are being checked.
  - *match profile* — The discovered hardware details on the host are being compared against known profiles.
  - *ready* — The host is available to be consumed.
  - *provisioning* — An image is being written to the host's disk(s).
  - *provisioning error* — The image could not be written to the host.
  - *provisioned* — An image has been completely written to the host's disk(s).
  - *externally provisioned* — Metal³ does not manage the image on the host.
  - *deprovisioning* — The image is being wiped from the host's disk(s).
  - *inspecting* — The hardware details for the host are being collected by an agent.
  - *power management error* — An error was found while trying to power the host either on or off.
- *id* — The unique identifier for the service in the underlying provisioning tool.
- *image* — The image most recently provisioned to the host.
- *rootDeviceHints* — The root device selection instructions used for the most recent provisioning operation.

### 1.1.3. BareMetalHost Example

The following is a complete example from a running cluster of a *BareMetalHost* resource (in YAML), it includes its specification and status sections:

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  creationTimestamp: "2019-09-20T06:33:35Z"
  finalizers:
  - baremetalhost.metal3.io
  generation: 2
  name: bmo-master-0
  namespace: bmo-project
  resourceVersion: "22642"
  selfLink: /apis/metal3.io/v1alpha1/namespaces/bmo-project/baremetalhosts/bmo-master-
0
```

```
  uid: 92b2f77a-db70-11e9-9db1-525400764849
spec:
  bmc:
    address: ipmi://10.10.57.19
    credentialsName: bmo-master-0-bmc-secret
  bootMACAddress: 98:03:9b:61:80:48
  consumerRef:
    apiVersion: machine.openshift.io/v1beta1
    kind: Machine
    name: bmo-master-0
    namespace: bmo-project
  externallyProvisioned: true
  hardwareProfile: default
  image:
    checksum: http://172.16.1.100/images/myOSv1/myOS.qcow2.md5sum
    url: http://172.16.1.100/images/myOSv1/myOS.qcow2
  online: true
  userData:
    name: bmo-master-user-data
    namespace: bmo-project
  networkData:
    name: bmo-master-network-data
    namespace: bmo-project
  metaData:
    name: bmo-master-meta-data
    namespace: bmo-project
status:
  errorMessage: ""
  goodCredentials:
    credentials:
      name: bmo-master-0-bmc-secret
      namespace: bmo-project
    credentialsVersion: "5562"
  hardware:
    cpu:
      arch: x86_64
      clockMegahertz: 2000
      count: 40
      flags: []
      model: Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz
    firmware:
      bios:
        date: 12/17/2018
        vendor: Dell Inc.
        version: 1.6.13
    hostname: bmo-master-0.localdomain
    nics:
    - ip: 172.22.135.105
      mac: "00:00:00:00:00:00"
      model: unknown
      name: eno1
```

```
      pxe: true
      speedGbps: 25
      vlanId: 0
    ramMebibytes: 0
    storage: []
    systemVendor:
      manufacturer: Dell Inc.
      productName: PowerEdge r460
      serialNumber: ""
  hardwareProfile: ""
  lastUpdated: "2019-09-20T07:03:23Z"
  operationalStatus: OK
  poweredOn: true
  provisioning:
    ID: a4438010-3fc6-4c5c-b570-900bbe85da57
    image:
      checksum: ""
      url: ""
    state: externally provisioned
  triedCredentials:
    credentials:
      name: bmo-master-0-bmc-secret
      namespace: bmo-project
    credentialsVersion: "5562"
```

And here is the secret `bmo-master-0-bmc-secret` holding the host's BMC credentials, base64 encoded:

```
$echo -n 'admin' | base64
YWRtaW4=

$echo -n 'password' | base64
cGFzc3dvcmQ=
```

Copy the above base64 encoded username and password pair and paste it into the yaml as mentioned below.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: bmo-master-0-bmc-secret
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
```

## 1.2. Triggering Provisioning

Several conditions must be met in order to initiate provisioning.

1. The host `spec.image.url` field must contain a URL for a valid image file that is visible from within the cluster and from the host receiving the image.
2. The host must have `online` set to `true` so that the operator will keep the host powered on.
3. The host must have all of the BMC details.

To initiate deprovisioning, clear the image URL from the host spec.

## 1.3. Unmanaged Hosts

Hosts created without BMC details will be left in the `unmanaged` state until the details are provided. Unmanaged hosts cannot be provisioned and their power state is undefined.

## 1.4. Pausing reconciliation

It is possible to pause the reconciliation of a BareMetalHost object by adding an annotation `baremetalhost.metal3.io/paused`. Metal³ provider sets the value of this annotation as `metal3.io/capm3` when the cluster to which the BareMetalHost belongs, is paused and removes it when the cluster is not paused. If you want to pause the reconciliation of BareMetalHost you can put any value on this annotation other than `metal3.io/capm3`. Please make sure that you remove the annotation only if the value of the annotation is not `metal3.io/capm3`, but another value that you have provided*. Removing the annotation will enable the reconciliation again.

# 2. Setup Development Environment

## 2.1. With minikube

1. Install and launch minikube
2. Create a namespace to host the operator

```
kubectl create namespace metal3
```

3. Install operator in the cluster

```
eval $(go env)
mkdir -p $GOPATH/src/github.com/metal3-io
cd $GOPATH/src/github.com/metal3-io
git clone https://github.com/metal3-io/baremetal-operator.git
cd baremetal-operator
kustomize build config/default | kubectl apply -f
```

4. Or Launch the operator locally

```
export OPERATOR_NAME=baremetal-operator
export DEPLOY_KERNEL_URL=http://172.22.0.1/images/ironic-python-agent.kernel
export DEPLOY_RAMDISK_URL=http://172.22.0.1/images/ironic-python-agent.initramfs
export IRONIC_ENDPOINT=http://localhost:6385/v1/
export IRONIC_INSPECTOR_ENDPOINT=http://localhost:5050/v1
make run
```

5. Create the CR

```
kubectl apply -f examples/example-host.yaml -n metal3
```

## 2.2. Running without Ironic

In environments where Ironic is not available, and the only real need is to be able to have some test data, use the test fixture provisioner instead of the real Ironic provisioner by passing `-test-mode` to the operator when launching it.

```
make run-test-mode
```

## 2.3. Running a local instance of Ironic

There is a script available that will run a set of containers locally using `podman` to stand up Ironic for development and testing.

See `tools/run_local_ironic.sh`.

Note that this script may need customizations to some of the `podman run` commands, to include environment variables that configure the containers for your environment. All ironic related environment variables are set by default if they are not passed through the environment.

The following environment variables can be passed to configure the ironic:

- HTTP_PORT - port used by httpd server (default 6180)
- PROVISIONING_IP - provisioning interface IP address to use for ironic, dnsmasq(dhcpd) and httpd (default 172.22.0.1)
- CLUSTER_PROVISIONING_IP - cluster provisioning interface IP address (default 172.22.0.2)
- PROVISIONING_CIDR - provisioning interface IP address CIDR (default 24)
- PROVISIONING_INTERFACE - interface to use for ironic, dnsmasq(dhcpd) and httpd (default ironicendpoint)
- CLUSTER_DHCP_RANGE - dhcp range to use for provisioning (default 172.22.0.10-172.22.0.100)
- DEPLOY_KERNEL_URL - the URL of the kernel to deploy ironic-python-agent

- DEPLOY_RAMDISK_URL - the URL of the ramdisk to deploy ironic-python-agent
- IRONIC_ENDPOINT - the endpoint of the ironic
- IRONIC_INSPECTOR_ENDPOINT - the endpoint of the ironic inspector
- CACHEURL - the URL of the cached images
- IRONIC_FAST_TRACK - whether to enable fast_track provisioning or not (default true)

## 2.4. Using Tilt for development

It is easy to use Tilt for BMO deployment. Once you have a local instance of Ironic running, just run

```
make tilt-up
```

and clean it with

```
make kind-reset
```

It is also possible to develop Baremetal Operator using Tilt with CAPM3. Please refer to the development setup guide of CAPM3 and specially the Baremetal Operator Integration

## 2.5. Using libvirt VMs with Ironic

In order to use VMs as hosts, they need to be connected to vbmc and the `bootMACAddress` field needs to be set to the MAC address of the network interface that will PXE boot.

For example:

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: worker-0
spec:
  online: true
  bmc:
    address: libvirt://192.168.122.1:6233/
    credentialsName: worker-0-bmc-secret
  bootMACAddress: 00:73:49:3a:76:8e
```

The `make-virt-host` utility can be used to generate a YAML file for registering a host. It takes as input the name of the `virsh` domain and produces as output the basic YAML to register that host properly, with the boot MAC address and BMC address filled in.

```
$ go run cmd/make-virt-host/main.go openshift_worker_1
---
apiVersion: v1
kind: Secret
metadata:
  name: openshift-worker-1-bmc-secret
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=

---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: openshift-worker-1
spec:
  online: true
  bmc:
    address: libvirt://192.168.122.1:6234/
    credentialsName: openshift-worker-1-bmc-secret
  bootMACAddress: 00:1a:74:74:e5:cf
```

The output can be passed directly to `kubectl apply` like this:

```
go run cmd/make-virt-host/main.go openshift_worker_1 | kubectl apply -f -
```

When the host is a *master*, include the `-consumer` and `-consumer-namespace` options to associate the host with the existing `Machine` object.

```
$ go run cmd/make-virt-host/main.go -consumer ostest-master-1 \
  -consumer-namespace openshift-machine-api  openshift_master_1
---
apiVersion: v1
kind: Secret
metadata:
  name: openshift-master-1-bmc-secret
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=

---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: openshift-master-1
spec:
  online: true
  bmc:
    address: libvirt://192.168.122.1:6231/
    credentialsName: openshift-master-1-bmc-secret
  bootMACAddress: 00:c9:a0:f2:e0:59
  consumerRef:
    name: ostest-master-1
    namespace: openshift-machine-api
```

## 2.6. Using Bare Metal Hosts

The `make-bm-worker` tool may be a more convenient way of creating YAML definitions for workers than editing the files directly.

```
$ go run cmd/make-bm-worker/main.go -address 1.2.3.4 \
  -password password -user admin worker-99
```

```
apiVersion: v1
kind: Secret
metadata:
  name: worker-99-bmc-secret
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=

---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: worker-99
spec:
  online: true
  bmc:
    address: 1.2.3.4
    credentialsName: worker-99-bmc-secret
    disableCertificateVerification: true
```

# 3. Running the tests

## 3.1. Setup

The user running the tests must have permission on the cluster to create CRDs. An example role binding setting for configuring the "developer" user is provided in test/e2e/role_binding.yaml

```
oc --as system:admin apply -f test/e2e/role_binding.yaml
```

### 3.1.1. Run the unit tests

The tests can be run via `make`

```
make test
```

Run linters test before pushing your commit

```
make lint
```

## 3.2. Using the Hack scripts

The repository contains a `hack` directory which has some very useful scripts for setting up

containerized testing environments. It also has Dockerfiles to allow you to generate images for your own use.

All the scripts accept the `CONTAINER_RUNTIME` environment variable with default of `podman`. This you can edit to `docker` as well.

### 3.2.1. Example of running `golint` in a Docker container

1. Start by setting the environment variable appropriately

```
export CONTAINER_RUNTIME=docker
```

2. From the operator parent dir, you can invoke the hack scripts

```
./hack/golint.sh
```

**Note** It's important to be in the operator parent dir because it contains the `Makefile` that is used in running the tests for the operator. Otherwise, you'll see the following error

```
sh: 0: Can't open /go/src/github.com/metal3-io/baremetal-operator/hack/golint.sh
```

3. Upon successful execution, you should see the following output. I already had all the images available from a previous run, you might see the images getting downloaded if you're running for the very first time.

```
+ IS_CONTAINER=false
+ CONTAINER_RUNTIME=docker
+ [ false != false ]
+ docker run --rm --env IS_CONTAINER=TRUE --volume
/home/noor/go/src/github.com/metal3-io/baremetal-
operator:/go/src/github.com/metal3-io/baremetal-operator:ro,z\
--entrypoint sh --workdir /go/src/github.com/metal3-io/baremetal-operator\
quay.io/metal3-io/golint:latest /go/src/github.com/metal3-io/baremetal-
operator/hack/golint.sh
+ IS_CONTAINER=TRUE
+ CONTAINER_RUNTIME=podman
+ [ TRUE != false ]
+ export XDG_CACHE_HOME=/tmp/.cache
+ make lint
which golint 2>&1 >/dev/null || make OPATH/bin/golint
find ./pkg ./cmd -type f -name \*.go  |grep -v zz_ | xargs -L1 golint
-set_exit_status
```

## 3.3. Getting the latest tooling

For testing without the hack scripts, make sure you install the latest development tools using `go get`, e-g

```
# go get golang.org/x/lint/golint
```

This ensures that the linters you have locally and the ones running in the CI are matched and helps avoid inconsistencies. Happy coding!

# 4. Running Baremetal Operator with or without Ironic

This document explains the deployment scenarios of deploying Bare Metal Operator(BMO) with or without Ironic as well as deploying only Ironic scenario.

**These are the deployment use cases in this document:**

1. Deploying baremetal-operator with Ironic.
2. Deploying baremetal-operator without Ironic.
3. Deploying only Ironic.

## 4.1. Current structure of baremetal-operator config directory

```
tree config/
config/
├── basic-auth
│   ├── default
│   │   ├── credentials.yaml
│   │   └── kustomization.yaml
│   └── tls
│       ├── credentials.yaml
│       └── kustomization.yaml
├── certmanager
│   ├── certificate.yaml
│   ├── kustomization.yaml
│   └── kustomizeconfig.yaml
├── crd
│   ├── bases
│   │   └── metal3.io_baremetalhosts.yaml
│   ├── kustomization.yaml
│   ├── kustomizeconfig.yaml
│   └── patches
```

```
│       ├── cainjection_in_baremetalhosts.yaml
│       └── webhook_in_baremetalhosts.yaml
├── default
│   ├── ironic.env
│   ├── kustomization.yaml
│   ├── manager_auth_proxy_patch.yaml
│   ├── manager_webhook_patch.yaml
│   └── webhookcainjection_patch.yaml
├── kustomization.yaml
├── manager
│   ├── kustomization.yaml
│   └── manager.yaml
├── namespace
│   ├── kustomization.yaml
│   └── namespace.yaml
├── prometheus
│   ├── kustomization.yaml
│   └── monitor.yaml
├── rbac
│   ├── auth_proxy_client_clusterrole.yaml
│   ├── auth_proxy_role_binding.yaml
│   ├── auth_proxy_role.yaml
│   ├── auth_proxy_service.yaml
│   ├── baremetalhost_editor_role.yaml
│   ├── baremetalhost_viewer_role.yaml
│   ├── kustomization.yaml
│   ├── leader_election_role_binding.yaml
│   ├── leader_election_role.yaml
│   ├── role_binding.yaml
│   └── role.yaml
├── render
│   └── capm3.yaml
├── samples
│   └── metal3.io_v1alpha1_baremetalhost.yaml
├── tls
│   ├── kustomization.yaml
│   └── tls_ca.yaml
└── webhook
    ├── kustomization.yaml
    ├── kustomizeconfig.yaml
    └── service.yaml
```

The `config` directory has one top level folder for deployment, namely `default` and it deploys only baremetal-operator through kustomization file calling `manager` folder. In addition, `basic-auth`, `certmanager`, `crd`, `namespace`, `prometheus`, `rbac`, `tls` and `webhook`folders have their own kustomization and yaml files.

## 4.2. Current structure of ironic-deployment directory

```
tree ironic-deployment/
ironic-deployment/
├── basic-auth
│   ├── default
│   │   ├── auth.yaml
│   │   └── kustomization.yaml
│   ├── ironic-auth-config-tpl
│   ├── ironic-inspector-auth-config-tpl
│   ├── ironic-rpc-auth-config-tpl
│   ├── keepalived
│   │   ├── auth.yaml
│   │   └── kustomization.yaml
│   └── tls
│       ├── default
│       │   ├── auth.yaml
│       │   └── kustomization.yaml
│       └── keepalived
│           ├── auth.yaml
│           └── kustomization.yaml
├── default
│   ├── ironic_bmo_configmap.env
│   └── kustomization.yaml
├── ironic
│   ├── ironic.yaml
│   └── kustomization.yaml
├── keepalived
│   ├── ironic_bmo_configmap.env
│   ├── keepalived_patch.yaml
│   └── kustomization.yaml
└── tls
    ├── default
    │   ├── kustomization.yaml
    │   └── tls.yaml
    └── keepalived
        ├── kustomization.yaml
        └── tls.yaml
```

Ironic-deployment folder has three top level folder for deployments, namely `default`, `ironic` and `keepalived`. `default` and `ironic` deploy only ironic, `keepalived` deploys the ironic with keepalived. As the name implies, `keepalived/keepalived_patch.yaml` patches the default image URL through kustomization. `tls` and `basic-auth` folders contain deployment files to add TLS and Basic Auth support between baremetal-operator and ironic.

## 4.3. Deployment commands

There is a useful deployment script that configures and deploys BareMetal Operator and Ironic. It

requires some variables :

- IRONIC_HOST : domain name for Ironic and inspector
- IRONIC_HOST_IP : IP on which Ironic and inspector are listening

In addition you can configure the following variables. They are **optional**. If you leave them unset, then passwords and certificates will be generated for you.

- KUBECTL_ARGS : Additional arguments to kubectl apply
- IRONIC_USERNAME : username for ironic
- IRONIC_PASSWORD : password for ironic
- IRONIC_INSPECTOR_USERNAME : username for inspector
- IRONIC_INSPECTOR_PASSWORD : password for inspector
- IRONIC_CACERT_FILE : CA certificate path for ironic
- IRONIC_CAKEY_FILE : CA certificate key path, unneeded if ironic certificates exist
- IRONIC_CERT_FILE : Ironic certificate path
- IRONIC_KEY_FILE : Ironic certificate key path
- IRONIC_INSPECTOR_CERT_FILE : Inspector certificate path
- IRONIC_INSPECTOR_KEY_FILE : Inspector certificate key path
- IRONIC_INSPECTOR_CACERT_FILE : CA certificate path for inspector, defaults to IRONIC_CACERT_FILE
- IRONIC_INSPECTOR_CAKEY_FILE : CA certificate key path, unneeded if inspector certificates exist

Then run :

```
./tools/deploy.sh <deploy-BMO> <deploy-Ironic> <deploy-TLS> <deploy-Basic-Auth>
<deploy-Keepalived>
```

- `deploy-BMO` : deploy BareMetal Operator : "true" or "false"
- `deploy-Ironic` : deploy Ironic : "true" or "false"
- `deploy-TLS` : deploy with TLS enabled : "true" or "false"
- `deploy-Basic-Auth` : deploy with Basic Auth enabled : "true" or "false"
- `deploy-Keepalived` : deploy with Keepalived for ironic : "true" or "false"

This will deploy BMO and / or Ironic with the proper configuration.

## 4.4. Useful tips

It is worth mentioning some tips for when the different configurations are useful as well. For example:

1. Only BMO is deployed, in a case when Ironic is already running, e.g. as part of Cluster API Provider Metal3 (CAPM3) when a successful pivoting state was met and ironic being deployed.
2. BMO and Ironic are deployed together, in a case when CAPM3 is not used and baremetal-operator and ironic containers to be deployed together.
3. Only Ironic is deployed, in a case when BMO is deployed as part of CAPM3 and only Ironic setup is sufficient, e.g. clusterctl provided by Cluster API(CAPI) deploys BMO, so that it can take care of moving the BaremetalHost during the pivoting.

**Important Note** When the baremetal-operator is deployed through metal3-dev-env, baremetal-operator container inherits the following environment variables through configmap:

```
$PROVISIONING_IP
$PROVISIONING_CIDR
$PROVISIONING_INTERFACE
```

In case you are deploying baremetal-operator locally, make sure to populate and export these environment variables before deploying.

# 5. Authenticating to Ironic

Because hosts under the control of Metal³ need to contact the Ironic and Ironic Inspector APIs during inspection and provisioning, it is highly advisable to require authentication on those APIs, since the provisioned hosts running user workloads will remain connected to the provisioning network.

## 5.1. Configuration

The `baremetal-operator` supports connecting to Ironic and Ironic Inspector configured with the following `auth_strategy` modes:

- `noauth` (no authentication)
- `http_basic` (HTTP Basic access authentication)

Note that Keystone authentication methods are not yet supported.

Authentication configuration is read from the filesystem, beginning at the root directory specified in the environment variable `METAL3_AUTH_ROOT_DIR`. If this variable is empty or not specified, the default is `/opt/metal3/auth`.

Within the root directory there are separate subdirectories, `ironic` for Ironic client configuration, and `ironic-inspector` for Ironic Inspector client configuration. (This allows the data to be populated from separate secrets when deploying in Kubernetes.)

### 5.1.1. `noauth`

This is the default, and will be chosen if the auth root directory does not exist. In this mode, the

baremetal-operator does not attempt to do any authentication against the Ironic APIs.

### 5.1.2. `http_basic`

This mode is configured by files in each authentication subdirectory named `username` and `password`, and containing the Basic auth username and password, respectively.