Resource Configuration

1 - Environments

Environments are a custom resource provided by the Metal Controller Manager. An environment is a codified description of what should be returned by the PXE server when a physical server attempts to PXE boot.

Especially important in the environment types are the kernel args. From here, one can tweak the IP to the metadata server as well as various other kernel options that Talos and/or the Linux kernel supports.

Environments can be supplied to a given server either at the Server or the ServerClass level. The hierarchy from most to least respected is:

- `.spec.environmentRef` provided at `Server` level
- `.spec.environmentRef` provided at `ServerClass` level
- `"default"` `Environment` created automatically and modified by an administrator

A sample environment definition looks like this:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: Environment
metadata:
  name: default
spec:
  kernel:
    url: "https://github.com/talos-systems/talos/releases/download/v0.14.0/vmlinuz-amd64"
    sha512: ""
    args:
      - console=tty0
      - console=ttyS1,115200n8
      - consoleblank=0
      - earlyprintk=ttyS1,115200n8
      - ima_appraise=fix
      - ima_hash=sha512
      - ima_template=ima-ng
      - init_on_alloc=1
      - initrd=initramfs.xz
      - nvme_core.io_timeout=4294967295
      - printk.devkmsg=on
      - pti=on
      - random.trust_cpu=on
      - slab_nomerge=
      - talos.platform=metal
  initrd:
    url: "https://github.com/talos-systems/talos/releases/download/v0.14.0/initramfs-amd64.xz"
    sha512: ""
```

Example of overriding `"default"` `Environment` at the `Server` level:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: Server
...
spec:
  environmentRef:
    namespace: default
    name: boot
  ...
```

Example of overriding `"default"` `Environment` at the `ServerClass` level:

```
apiVersion: metal.sidero.dev/v1alpha2
kind: ServerClass
...
spec:
  environmentRef:
    namespace: default
    name: boot
  ...
```

2 - Servers

Servers are the basic resource of bare metal in the Metal Controller Manager. These are created by PXE booting the servers and allowing them to send a registration request to the management plane.

An example server may look like the following:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: Server
metadata:
  name: 00000000-0000-0000-0000-d05099d333e0
  labels:
    common-label: "true"
    zone: east
    environment: test
spec:
  accepted: false
  configPatches:
    - op: replace
      path: /cluster/network/cni
      value:
        name: custom
        urls:
          - http://192.168.1.199/assets/cilium.yaml
  hardware:
    system:
      manufacturer: Dell Inc.
      productName: PowerEdge R630
      serialNumber: 790H8D2
    compute:
      totalCoreCount: 8
      totalThreadCount: 16
      processorCount: 1
      processors:
        - manufacturer: Intel
          productName: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
          speed: 2400
          coreCount: 8
          threadCount: 16
    memory:
      totalSize: 32 GB
      moduleCount: 2
      modules:
        - manufacturer: 002C00B3002C
          productName: 18ASF2G72PDZ-2G3B1
          serialNumber: 12BDC045
          type: LPDDR3
          size: 16384
          speed: 2400
        - manufacturer: 002C00B3002C
          productName: 18ASF2G72PDZ-2G3B1
          serialNumber: 12BDBF5D
          type: LPDDR3
          size: 16384
          speed: 2400
    storage:
      totalSize: 1116 GB
      deviceCount: 1
      devices:
        - productName: PERC H730 Mini
          type: HDD
          name: sda
          deviceName: /dev/sda
          size: 1199101181952
          wwid: naa.61866da055de070028d8e83307cc6df2
    network:
      interfaceCount: 2
      interfaces:
        - index: 1
          name: lo
          flags: up|loopback
          mtu: 65536
          mac: ""
          addresses:
            - 127.0.0.1/8
            - ::1/128
        - index: 2
          name: enp3s0
          flags: up|broadcast|multicast
          mtu: 1500
          mac: "40:8d:5c:86:5a:14"
          addresses:
            - 192.168.2.8/24
            - fe80::dcb3:295c:755b:91bb/64
```

Installation Disk

An installation disk is required by Talos on bare metal. This can be specified in a `configPatch`:

```
apiVersion: metal.sidero.dev/v1alpha2
kind: Server
...
spec:
  accepted: false
  configPatches:
    - op: replace
      path: /machine/install/disk
      value: /dev/sda
```

The install disk patch can also be set on the `ServerClass`:

```
apiVersion: metal.sidero.dev/v1alpha2
kind: ServerClass
...
spec:
  configPatches:
    - op: replace
      path: /machine/install/disk
      value: /dev/sda
```

Server Acceptance

In order for a server to be eligible for consideration, it *must* be `accepted`. This is an important separation point which all `Server`s must pass. Before a `Server` is accepted, no write action will be performed against it. Thus, it is safe for a computer to be added to a network on which Sidero is operating. Sidero will never write to or wipe any disk on a computer which is not marked as `accepted`.

This can be tedious for systems in which all attached computers should be considered to be under the control of Sidero. Thus, you may also choose to automatically accept any machine into Sidero on its discovery. Please keep in mind that this means that any newly-connected computer **WILL BE WIPED** automatically. You can enable auto-acceptance by passing the `--auto-accept-servers=true` flag to `sidero-controller-manager`.

Once accepted, a server will be reset (all disks wiped) and then made available to Sidero.

You should never change an accepted `Server` to be *not* accepted while it is in use. Because servers which are not accepted will not be modified, if a server which *was* accepted is changed to *not* accepted, the disk will *not* be wiped upon its exit.

IPMI

Sidero can use IPMI information to control `Server` power state, reboot servers and set boot order.

IPMI information will be, by default, setup automatically if possible as part of the acceptance process. In this design, a “sidero” user will be added to the IPMI user list and a randomly generated password will be issued. This information is then squirreled away in a Kubernetes secret in the `sidero-system` namespace, with a name format of `<server-uuid>-bmc`. Users wishing to turn off this feature can pass the `--auto-bmc-setup=false` flag to `sidero-controller-manager`, e.g. using `export SIDERO_CONTROLLER_MANAGER_AUTO_BMC_SETUP=false` during installation.

IPMI connection information can also be set manually in the `Server` spec after initial registration:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: Server
...
spec:
  bmc:
    endpoint: 10.0.0.25
    user: admin
    pass: password
```

If IPMI information is set, server boot order might be set to boot from disk, then network, Sidero will switch servers to PXE boot once that is required.

Without IPMI info, Sidero can still register servers, wipe them and provision clusters, but Sidero won't be able to reboot servers once they are removed from the cluster. **If IPMI info is not set, servers should be configured to boot first from network, then from disk.**

Sidero can also fetch IPMI credentials via the `Secret` reference:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: Server
...
spec:
  bmc:
    endpoint: 10.0.0.25
    userFrom:
      secretKeyRef:
        name: ipmi-credentials
        key: username
    passFrom:
      secretKeyRef:
        name: ipmi-credentials
        key: password
```

As the `Server` resource is not namespaced, `Secret` should be created in the `default` namespace.

Other Settings

`cordoned`

If `cordoned` is set to `true`, `Server` gets excluded from any `ServerClass` it might match based on qualifiers. This means that the `Server` will not be allocated automatically.

`Server` might be `cordoned` to temporarily take it out of the `ServerClass` to perform for example hardware maintenance.

```yaml
apiVersion: metal.sidero.dev/v1alpha1
kind: Server
...
spec:
  cordoned: true
```

`pxeBootAlways`

`Server` might be forced to boot from the network even if the OS is already installed with `pxeBootAlways: true`:

```yaml
apiVersion: metal.sidero.dev/v1alpha1
kind: Server
...
spec:
  pxeBootAlways: true
```

`bootFromDiskMethod`

The method to exit iPXE network boot to force boot from disk can be configured for the `Server`:

```yaml
apiVersion: metal.sidero.dev/v1alpha1
kind: Server
...
spec:
  bootFromDiskMethod: ipxe-sanboot
```

Valid values are:

- `ipxe-exit`
- `http-404`
- `ipxe-sanboot`

If not set, the `ServerClass.spec.bootFromDiskMethod` value is used with the fallback to the default boot from disk method (`SIDERO_CONTROLLER_MANAGER_BOOT_FROM_DISK_METHOD`).

3 - Server Classes

Server classes are a way to group distinct server resources. The `qualifiers` and `selector` keys allow the administrator to specify criteria upon which to group these servers. If both of these keys are missing, the server class matches all servers that it is watching. If both of these keys define requirements, these requirements are combined (logical `AND`).

`selector`

`selector` groups server resources by their labels. The Kubernetes documentation has more information on how to use this field.

`qualifiers`

A list of hardware criteria, where each entry in the list is interpreted as a logical `OR`. All criteria inside each entry is interpreted as a logical `AND`. Qualifiers that are not specified are not evaluated.

An example:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: ServerClass
metadata:
  name: serverclass-sample
spec:
  selector:
    matchLabels:
      common-label: "true"
    matchExpressions:
      - key: zone
        operator: In
        values:
          - central
          - east
      - key: environment
        operator: NotIn
        values:
          - prod
  qualifiers:
    hardware:
      - system:
          manufacturer: Dell Inc.
        compute:
          processors:
            - manufacturer: Advanced Micro Devices, Inc.
              productName: AMD Ryzen 7 2700X Eight-Core Processor
      - compute:
          processors:
            - manufacturer: "Intel(R) Corporation"
              productName: "Intel(R) Atom(TM) CPU C3558 @ 2.20GHz"
        memory:
          totalSize: "8 GB"
```

Servers would only be added to the above class if they:

- have the label `common-label` with value `true`
- *AND* match the `matchExpressions`
- *AND* match either 1 of the following criteria:
  - has a system manufactured by `Dell Inc.` *AND* has at least 1 processor that is an `AMD Ryzen 7 2700X Eight-Core Processor`
  - has at least 1 processor that is an `Intel(R) Atom(TM) CPU C3558 @ 2.20GHz` *AND* has exactly 8 GB of total memory

Additionally, Sidero automatically creates and maintains a server class called `"any"` that includes all (accepted) servers. Attempts to add qualifiers to it will be reverted.

`configPatches`

Server configs of servers matching a server class can be updated by using the `configPatches` section of the custom resource. See patching for more information on how this works.

An example of settings the default install disk for all servers matching a server class:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: ServerClass
...
spec:
  configPatches:
    - op: replace
      path: /machine/install/disk
      value: /dev/sda
```

Other Settings

`environmentRef`

Servers from a `ServerClass` can be set to use the specific `Environment` by linking the `Environment` from the `ServerClass` :

```yaml
apiVersion: metal.sidero.dev/v1alpha1
kind: ServerClass
...
spec:
  environmentRef:
    name: production-env
```

`bootFromDiskMethod`

The method to exit iPXE network boot to force boot from disk can be configured for all `Server` resources belonging to the `ServerClass` :

```yaml
apiVersion: metal.sidero.dev/v1alpha1
kind: ServerClass
...
spec:
  bootFromDiskMethod: ipxe-sanboot
```

Valid values are:

- `ipxe-exit`
- `http-404`
- `ipxe-sanboot`

If not set, the default boot from disk method is used (`SIDERO_CONTROLLER_MANAGER_BOOT_FROM_DISK_METHOD`).

4 - Metadata

The Sidero controller manager manages the Machine metadata. In terms of Talos (the OS on which the Kubernetes cluster is formed), this is the "machine config", which is used during the automated installation.

Talos Machine Configuration

The configuration of each machine is constructed from a number of sources:

- The `TalosControlPlane` custom resource for control plane nodes.
- The `TalosConfigTemplate` custom resource.
- The `ServerClass` which was used to select the `Server` into the `Cluster` .
- Any `Server` -specific patches.

An example usage of setting a virtual IP for the control plane nodes and adding extra `node-labels` to nodes is shown below:

> Note: because of the way JSON patches work the interface setting also needs to be set in `TalosControlPlane` when defining a Virtual IP. This experience is not ideal, but will be addressed in a future release.

TalosControlPlane custom resource:

```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha3
kind: TalosControlPlane
metadata:
  name: workload-cluster
  namespace: default
spec:
  controlPlaneConfig:
    controlplane:
      configPatches:
      - op: add
        path: /machine/network
        value:
          interfaces:
          - interface: eth0
            dhcp: true
            vip:
              ip: 172.16.200.52
      generateType: controlplane
      talosVersion: v0.13
    init:
      configPatches:
      - op: add
        path: /machine/network
        value:
          interfaces:
          - interface: eth0
            dhcp: true
            vip:
              ip: 172.16.200.52
      generateType: init
      talosVersion: v0.13
  infrastructureTemplate:
    apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
    kind: MetalMachineTemplate
    name: workload-cluster
  replicas: 3
  version: v1.23.0
```

TalosConfigTemplate custom resource:

```yaml
---
apiVersion: bootstrap.cluster.x-k8s.io/v1alpha3
kind: TalosConfigTemplate
metadata:
  name: workload-cluster
  namespace: default
spec:
  template:
    spec:
      generateType: join
      talosVersion: v0.13
      configPatches:
      - op: add
        path: /machine/kubelet
        value:
          extraArgs:
            node-labels:
              talos.dev/part-of: cluster/workload-cluster
```

and finally in the control plane `ServerClass` custom resource we augment the network information for other interfaces:

```yaml
---
apiVersion: metal.sidero.dev/v1alpha2
kind: ServerClass
metadata:
  name: cp.small.x86
spec:
  configPatches:
  - op: replace
    path: /machine/install/disk
    value: /dev/nvme0n1
  - op: add
    path: /machine/install/extraKernelArgs
    value:
    - console=tty0
    - console=ttyS1,115200n8
  - op: add
    path: /machine/network/interfaces/-
    value:
      interface: eth1
      dhcp: true
  qualifiers:
    - system:
        manufacturer: Supermicro
      compute:
        processors:
          - productName: Intel(R) Xeon(R) E-2124G CPU @ 3.40GHz
      memory:
        totalSize: 8 GB
  selector:
    matchLabels:
      metal.sidero.dev/serverclass: cp.small.x86
```

the workload `ServerClass` defines the complete networking config

```yaml
---
apiVersion: metal.sidero.dev/v1alpha2
kind: ServerClass
metadata:
  name: general.medium.x86
spec:
  configPatches:
  - op: replace
    path: /machine/install/disk
    value: /dev/nvme1n1
  - op: add
    path: /machine/install/extraKernelArgs
    value:
    - console=tty0
    - console=ttyS1,115200n8
  - op: add
    path: /machine/network
    value:
      interfaces:
      - interface: eth0
        dhcp: true
      - interface: eth1
        dhcp: true
  qualifiers:
    - system:
        manufacturer: Supermicro
      compute:
        processors:
          - productName: Intel(R) Xeon(R) E-2136 CPU @ 3.30GHz
      memory:
        totalSize: 16 GB
  selector:
    matchLabels:
      metal.sidero.dev/serverclass: general.medium.x86
```

The base template is constructed from the Talos bootstrap provider, using data from the associated `TalosControlPlane` and `TalosConfigTemplate` manifest. Then, any configuration patches are applied from the `ServerClass` and `Server`.

These patches take the form of an RFC 6902 JSON (or YAML) patch. An example of the use of this patch method can be found in Patching Guide.

Also note that while a `Server` can be a member of any number of `ServerClass` es, only the `ServerClass` which is used to select the `Server` into the `Cluster` will be used for the generation of the configuration of the `Machine`. In this way, `Servers` may have a number of different configuration patch sets based on which `Cluster` they are in at any given time.