Overview

1 - Introduction

Sidero (“Iron” in Greek) is a project created by the Sidero Labs team. Sidero Metal provides lightweight, composable tools that can be used to create bare-metal Talos Linux + Kubernetes clusters. These tools are built around the Cluster API project.

Because of the design of Cluster API, there is inherently a “chicken and egg” problem: you need an existing Kubernetes cluster in order to provision the management plane, that can then provision more clusters. The initial management plane cluster that runs the Sidero Metal provider does not need to be based on Talos Linux - although it is recommended for security and stability reasons. The Getting Started guide will walk you through installing Sidero Metal either on an existing cluster, or by quickly creating a docker based cluster used to bootstrap the process.

Overview

Sidero Metal is currently made up of two components:

- Metal Controller Manager: Provides custom resources and controllers for managing the lifecycle of metal machines, iPXE server, metadata service, and gRPC API service
- Cluster API Provider Sidero (CAPS): A Cluster API infrastructure provider that makes use of the pieces above to spin up Kubernetes clusters

Sidero Metal also needs these co-requisites in order to be useful:

- Cluster API
- Cluster API Control Plane Provider Talos
- Cluster API Bootstrap Provider Talos

All components mentioned above can be installed using Cluster API’s `clusterctl` tool. See the Getting Started for more details.

2 - What's New

New API Version for `metal.sidero.dev` Resources

Resources under `metal.sidero.dev` (`Server`, `ServerClass`, `Environment`) now have a new version `v1alpha2`. Old version `v1alpha1` is still supported, but it is recommended to update templates to use the new resource version.

`Server` Changes

Hardware information was restructured and extended when compared with `v1alpha1`:

- `.spec.systemInformation -> .spec.hardware.system`
- `.spec.cpu -> .spec.hardware.compute.processors[]`

`ServerClass` Changes

- `.spec.qualifiers.systemInformation -> .spec.qualifiers.system`
- `.spec.qualifiers.cpu -> .spec.qualifiers.hardware.compute.processors[]`

Metadata Server

Sidero Metadata Server no longer depends on the version of Talos machinery library it is built with. Sidero should be able to process machine config for future versions of Talos.

Sidero Agent

Sidero Agent now runs DHCP client in the userland, on the link which was used to PXE boot the machine. This allows to run Sidero Agent on the machine with several autoconfigured network interfaces, when one of them is used for the management network.

DHCP Proxy

Sidero Controller Manager now includes DHCP proxy which augments DHCP response with additional PXE boot options. When enabled, DHCP server in the environment only handles IP allocation and network configuration, while DHCP proxy provides PXE boot information automatically based on the architecture and boot method.

3 - Installation

To install Sidero and the other Talos providers, simply issue:

```
clusterctl init -b talos -c talos -i sidero
```

Sidero supports several variables to configure the installation, these variables can be set either as environment variables or as variables in the `clusterctl` configuration:

- `SIDERO_CONTROLLER_MANAGER_HOST_NETWORK` (`false`): run `sidero-controller-manager` on host network
- `SIDERO_CONTROLLER_MANAGER_DEPLOYMENT_STRATEGY` (`RollingUpdate`): strategy to use when updating `sidero-controller-manager`, use `Recreate` when using a single node and `SIDERO_CONTROLLER_MANAGER_HOST_NETWORK` is `true`
- `SIDERO_CONTROLLER_MANAGER_API_ENDPOINT` (empty): specifies the IP address controller manager API service can be reached on, defaults to the node IP (TCP)
- `SIDERO_CONTROLLER_MANAGER_API_PORT` (8081): specifies the port controller manager can be reached on
- `SIDERO_CONTROLLER_MANAGER_CONTAINER_API_PORT` (8081): specifies the controller manager internal container port
- `SIDERO_CONTROLLER_MANAGER_SIDEROLINK_ENDPOINT` (empty): specifies the IP address SideroLink Wireguard service can be reached on, defaults to the node IP (UDP)
- `SIDERO_CONTROLLER_MANAGER_SIDEROLINK_PORT` (51821): specifies the port SideroLink Wireguard service can be reached on
- `SIDERO_CONTROLLER_MANAGER_EXTRA_AGENT_KERNEL_ARGS` (empty): specifies additional Linux kernel arguments for the Sidero agent (for example, different console settings)
- `SIDERO_CONTROLLER_MANAGER_AUTO_ACCEPT_SERVERS` (`false`): automatically accept discovered servers, by default `.spec.accepted` should be changed to `true` to accept the server
- `SIDERO_CONTROLLER_MANAGER_AUTO_BMC_SETUP` (`true`): automatically attempt to configure the BMC with a `sidero` user that will be used for all IPMI tasks.
- `SIDERO_CONTROLLER_MANAGER_INSECURE_WIPE` (`true`): wipe only the first megabyte of each disk on the server, otherwise wipe the full disk
- `SIDERO_CONTROLLER_MANAGER_SERVER_REBOOT_TIMEOUT` (`20m`): timeout for the server reboot (how long it might take for the server to be rebooted before Sidero retries an IPMI reboot operation)
- `SIDERO_CONTROLLER_MANAGER_IPMI_PXE_METHOD` (`uefi`): IPMI boot from PXE method: `uefi` for UEFI boot or `bios` for BIOS boot
- `SIDERO_CONTROLLER_MANAGER_BOOT_FROM_DISK_METHOD` (`ipxe-exit`): configures the way Sidero forces server to boot from disk when server hits iPXE server after initial install: `ipxe-exit` returns iPXE script with `exit` command, `http-404` returns HTTP 404 Not Found error, `ipxe-sanboot` uses iPXE `sanboot` command to boot from the first hard disk (can be also configured on `ServerClass` / `Server` method)
- `SIDERO_CONTROLLER_MANAGER_DISABLE_DHCP_PROXY` (`false`): disable DHCP Proxy service (enabled by default)
- `SIDERO_CONTROLLER_MANAGER_EVENTS_NEGATIVE_ADDRESS_FILTER` (empty): negative filter for reported machine addresses (e.g. `10.0.0.0/8` won't publish any `10.x` addresses to the `MetalMachine` status)

Sidero provides four endpoints which should be made available to the infrastructure:

- UDP port 67 for the proxy DHCP service (providing PXE boot information to the nodes, but no IPAM)
- TCP port 8081 which provides combined iPXE, metadata and gRPC service (external endpoint should be specified as `SIDERO_CONTROLLER_MANAGER_API_ENDPOINT` and `SIDERO_CONTROLLER_MANAGER_API_PORT`)
- UDP port 69 for the TFTP service (DHCP server should point the nodes to PXE boot from that IP)
- UDP port 51821 for the SideroLink Wireguard service (external endpoint should be specified as `SIDERO_CONTROLLER_MANAGER_SIDEROLINK_ENDPOINT` and `SIDERO_CONTROLLER_MANAGER_SIDEROLINK_PORT`)

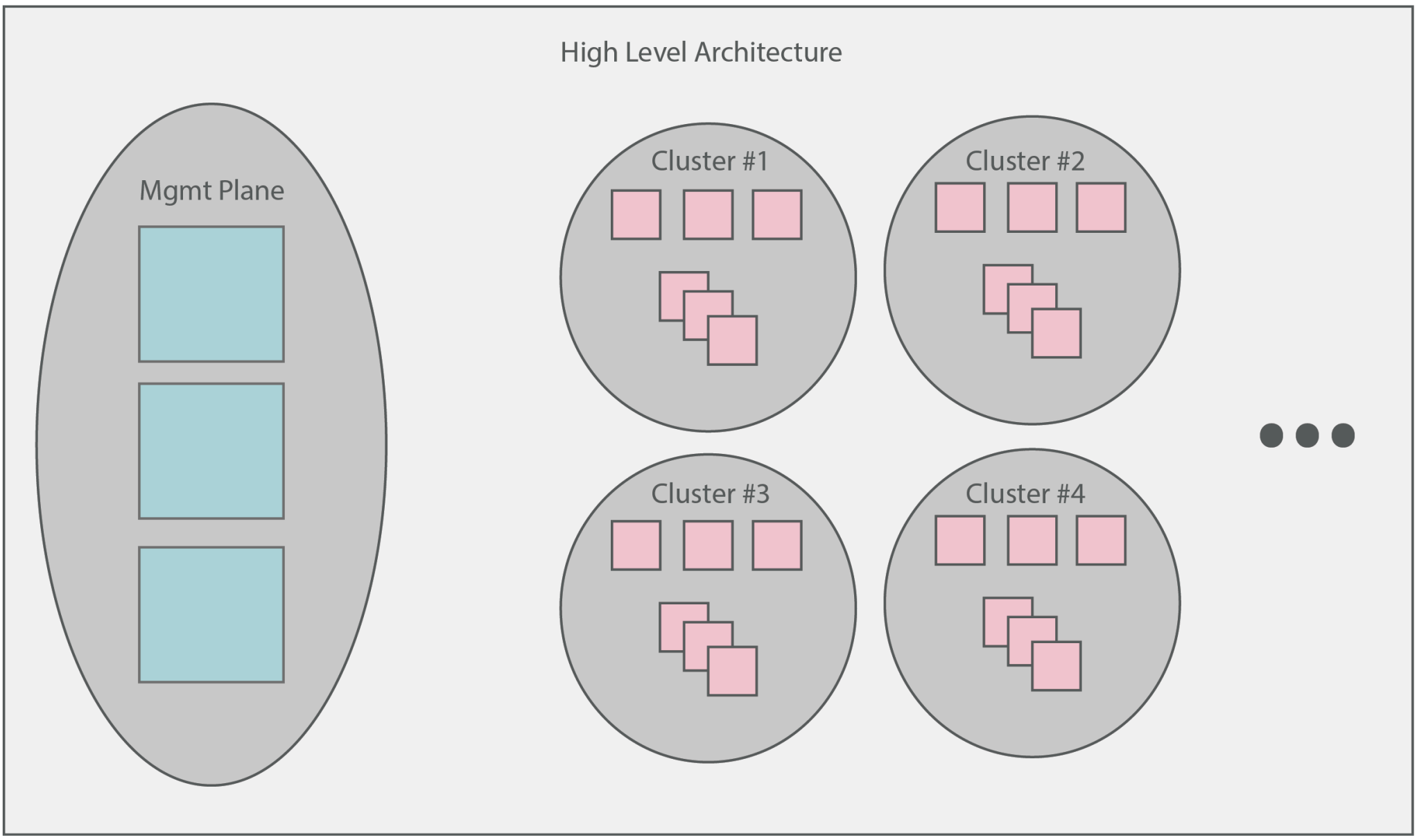
These endpoints could be exposed to the infrastructure using different strategies:

- running `sidero-controller-manager` on the host network.
- using Kubernetes load balancers (e.g. MetalLB), ingress controllers, etc.

> Note: If you want to run `sidero-controller-manager` on the host network using port different from `8081` you should set both `SIDERO_CONTROLLER_MANAGER_API_PORT` and `SIDERO_CONTROLLER_MANAGER_CONTAINER_API_PORT` to the same value.

4 - Architecture

The overarching architecture of Sidero centers around a “management plane”. This plane is expected to serve as a single interface upon which administrators can create, scale, upgrade, and delete Kubernetes clusters. At a high level view, the management plane + created clusters should look something like:



5 - SideroLink

SideroLink provides an overlay Wireguard point-to-point connection from every Talos machine to the Sidero. Sidero provisions each machine with a unique IPv6 address and Wireguard key for the SideroLink connection.

> Note: SideroLink is only supported with Talos >= 0.14.
>
> SideroLink doesn't provide a way for workload machines to communicate with each other, a connection is only point-to-point.

SideroLink connection is both encrypted and authenticated, so Sidero uses that to map data streams coming from the machines to a specific `ServerBinding`, `MetalMachine`, `Machine` and `Cluster`.

Talos node sends two streams over the SideroLink connection: kernel logs (dmesg) and Talos event stream. SideroLink is enabled automatically by Sidero when booting Talos.

Kernel Logs

Kernel logs (`dmesg`) are streamed in real time from the Talos nodes to the `sidero-controller-manager` over SideroLink connection. Log streaming starts when the kernel passes control to the `init` process, so kernel boot time logs will only be available when control is passed to the userland.

Logs can be accessed by accessing the logs of the `serverlogs` container of the `sidero-controller-manager` pod:

```
$ kubectl -n sidero-system logs deployment/sidero-controller-manager -c serverlogs -f
{"clock":8576583,"cluster":"management-cluster","facility":"user","machine":"management-cluster-cp-ddgsw","metal_machin
...
```

The format of the message is the following:

```
{
    "clock": 8576583,
    "cluster": "management-cluster",
    "facility": "user",
    "machine": "management-cluster-cp-ddgsw",
    "metal_machine": "management-cluster-cp-vrff4",
    "msg": "[talos] phase mountState (6/13): 1 tasks(s)\n",
    "namespace": "default",
    "priority": "warning",
    "seq": 665,
    "server_uuid": "6b121f82-24a8-4611-9d23-fa1a5ba564f0",
    "talos-level": "warn",
    "talos-time": "2022-02-11T12:42:02.74807823Z"
}
```

Kernel fields (see Linux documentation for details):

- `clock` is the kernel timestamp relative to the boot time
- `facility` of the message
- `msg` is the actual log message
- `seq` is the kernel log sequence
- `priority` is the message priority

Talos-added fields:

- `talos-level` is the translated `priority` into standard logging levels
- `talos-time` is the timestamp of the log message (accuracy of the timestamp depends on time sync)

Sidero-added fields:

- `server_uuid` is the `name` of the matching `Server` and `ServerBinding` resources

- `namespace` is the namespace of the `Cluster`, `MetalMachine` and `Machine`
- `cluster`, `metal_machine` and `machine` are the names of the matching `Cluster`, `MetalMachine` and `Machine` resources

It might be a good idea to send container logs to some log aggregation system and filter the logs for a cluster or a machine.

Quick filtering for a specific server:

```
kubectl -n sidero-system logs deployment/sidero-controller-manager -c serverlogs  | jq -R 'fromjson? | select(.server_u
```

Talos Events

Talos delivers system events over the SideroLink connection to the `sidero-link-manager` pod. These events can be accessed with `talosctl events` command. Events are mostly used to update `ServerBinding` / `MetalMachine` statuses, but they can be also seen in the logs of the `serverevents` container:

```
$ kubectl -n sidero-system logs deployment/sidero-controller-manager -c serverevents -f
{"level":"info","ts":1644853714.2700942,"caller":"events-manager/adapter.go:153","msg":"incoming event","component":"si
AddressEvent","server_uuid":"b4e677d9-b59b-4c1c-925a-f9d9ce049d79","cluster":"management-cluster","namespace":"default"
```

MetalMachine Conditions

Sidero updates the statuses of `ServerBinding` / `MetalMachine` resources based on the events received from Talos node:

- current addresses of the node
- statuses of machine configuration loading and validation, installation status

See Resources for details.

SideroLink State

State of the SideroLink connection is kept in the `ServerBinding` resource:

```
spec:
  siderolink:
    address: fdae:2859:5bb1:7a03:3ae3:be30:7ec4:4c09/64
    publicKey: XIBT49g9xCoBvyb/x36J+ASlQ4qaxXMG20ZgKbBbfE8=
```

Installation-wide SideroLink state is kept in the `siderolink` `Secret` resource:

```
$ kubectl get secrets siderolink -o yaml
apiVersion: v1
data:
  installation-id: QUtmZGFmVGJtUGVFcWp0RGMzT1BHSzlGcmlHTzdDQ0JCSU9aRzRSamdtWT0=
  private-key: ME05bHhBd3JwV0hDczhNbm1aR3RDL1ZjK0ZSUFM5UzQwd25IU00wQ3dHOD0=
...
```

Key `installation-id` is used to generate unique SideroLink IPv6 addresses, and `private-key` is the Wireguard key of Sidero.

6 - Resources

Sidero, the Talos bootstrap/controlplane providers, and Cluster API each provide several custom resources (CRDs) to Kubernetes. These CRDs are crucial to understanding the connections between each provider and in troubleshooting problems. It may also help to look at the cluster template to get an idea of the relationships between these.

Cluster API (CAPI)

It's worth defining the most basic resources that CAPI provides first, as they are related to several subsequent resources below.

`Cluster`

`Cluster` is the highest level CAPI resource. It allows users to specify things like network layout of the cluster, as well as contains references to the infrastructure and control plane resources that will be used to create the cluster.

`Machines`

`Machine` represents an infrastructure component hosting a Kubernetes node. Allows for specification of things like Kubernetes version, as well as contains reference to the infrastructure resource that relates to this machine.

`MachineDeployments`

`MachineDeployments` are similar to a `Deployment` and their relationship to `Pods` in Kubernetes primitives. A `MachineDeployment` allows for specification of a number of Machine replicas with a given specification.

Cluster API Bootstrap Provider Talos (CABPT)

`TalosConfigs`

The `TalosConfig` resource allows a user to specify the type (init, controlplane, join) for a given machine. The bootstrap provider will then generate a Talos machine configuration for that machine. This resource also provides the ability to pass a full, pre-generated machine configuration. Finally, users have the ability to pass `configPatches`, which are applied to edit a generate machine configuration with user-defined settings. The `TalosConfig` corresponds to the `bootstrap` sections of Machines, `MachineDeployments`, and the `controlPlaneConfig` section of `TalosControlPlanes`.

`TalosConfigTemplates`

`TalosConfigTemplates` are similar to the `TalosConfig` above, but used when specifying a bootstrap reference in a `MachineDeployment`.

Cluster API Control Plane Provider Talos (CACPPT)

`TalosControlPlanes`

The control plane provider presents a single CRD, the `TalosControlPlane`. This resource is similar to `MachineDeployments`, but is targeted exclusively for the Kubernetes control plane nodes. The `TalosControlPlane` allows for specification of the number of replicas, version of Kubernetes for the control plane nodes, references to the infrastructure resource to use (`infrastructureTemplate` section), as well as the configuration of the bootstrap data via the `controlPlaneConfig` section. This resource is referred to by the CAPI Cluster resource via the `controlPlaneRef` section.

Sidero

Cluster API Provider Sidero (CAPS)

MetalClusters

A `MetalCluster` is Sidero's view of the cluster resource. This resource allows users to define the control plane endpoint that corresponds to the Kubernetes API server. This resource corresponds to the `infrastructureRef` section of Cluster API's `Cluster` resource.

MetalMachines

A `MetalMachine` is Sidero's view of a machine. Allows for reference of a single server or a server class from which a physical server will be picked to bootstrap.

`MetalMachine` provides a set of statuses describing the state (available with SideroLink, requires Talos >= 0.14):

```
status:
  addresses:
    - address: 172.25.0.5
      type: InternalIP
    - address: pxe-2
      type: Hostname
  conditions:
    - lastTransitionTime: "2022-02-11T14:20:42Z"
      message: 'Get ... connection refused'
      reason: ProviderUpdateFailed
      severity: Warning
      status: "False"
      type: ProviderSet
    - lastTransitionTime: "2022-02-11T12:48:35Z"
      status: "True"
      type: TalosConfigLoaded
    - lastTransitionTime: "2022-02-11T12:48:35Z"
      status: "True"
      type: TalosConfigValidated
    - lastTransitionTime: "2022-02-11T12:48:35Z"
      status: "True"
      type: TalosInstalled
```

Statuses:

- `addresses` lists the current IP addresses and hostname of the node, `addresses` are updated when the node addresses are changed
- `conditions`:
  - `ProviderSet`: captures the moment infrastrucutre provider ID is set in the `Node` specification; depends on workload cluster control plane availability
  - `TalosConfigLoaded`: Talos successfully loaded machine configuration from Sidero; if this condition indicates a failure, check `sidero-controller-manager` logs
  - `TalosConfigValidated`: Talos successfully validated machine configuration; a failure in this condition indicates that the machine config is malformed
  - `TalosInstalled`: Talos was successfully installed to disk

MetalMachineTemplates

A `MetalMachineTemplate` is similar to a `MetalMachine` above, but serves as a template that is reused for resources like `MachineDeployments` or `TalosControlPlanes` that allocate multiple `Machines` at once.

ServerBindings

`ServerBindings` represent a one-to-one mapping between a Server resource and a `MetalMachine` resource. A `ServerBinding` is used internally to keep track of servers that are allocated to a Kubernetes cluster and used to make decisions on cleaning and returning servers to a `ServerClass` upon deallocation.

Metal Controller Manager

Environments

These define a desired deployment environment for Talos, including things like which kernel to use, kernel args to pass, and the initrd to use. Sidero allows you to define a default environment, as well as other environments that may be specific to a subset of nodes. Users can override the environment at the `ServerClass` or `Server` level, if you have requirements for different kernels or kernel parameters.

See the Environments section of our Configuration docs for examples and more detail.

Servers

These represent physical machines as resources in the management plane. These `Servers` are created when the physical machine PXE boots and completes a “discovery” process in which it registers with the management plane and provides SMBIOS information such as the CPU manufacturer and version, and memory information.

See the Servers section of our Configuration docs for examples and more detail.

ServerClasses

`ServerClasses` are a grouping of the `Servers` mentioned above, grouped to create classes of servers based on Memory, CPU or other attributes. These can be used to compose a bank of `Servers` that are eligible for provisioning.

See the ServerClasses section of our Configuration docs for examples and more detail.

Sidero Controller Manager

While the controller does not present unique CRDs within Kubernetes, it’s important to understand the metadata resources that are returned to physical servers during the boot process.

Metadata

The Sidero controller manager server may be familiar to you if you have used cloud environments previously. Using Talos machine configurations created by the Talos Cluster API bootstrap provider, along with patches specified by editing `Server` / `ServerClass` resources or `TalosConfig` / `TalosControlPlane` resources, metadata is returned to servers who query the controller manager at boot time.

See the Metadata section of our Configuration docs for examples and more detail.

7 - System Requirements

System Requirements

Most of the time, Sidero does very little, so it needs very few resources. However, since it is in charge of any number of workload clusters, it **should** be built with redundancy. It is also common, if the cluster is single-purpose, to combine the controlplane and worker node roles. Virtual machines are also perfectly well-suited for this role.

Minimum suggested dimensions:

- Node count: 3
- Node RAM: 4GB
- Node CPU: ARM64 or x86-64 class
- Node storage: 32GB storage on system disk