Guides

1 - Bootstrapping

A guide for bootstrapping Sidero management plane

Introduction

Imagine a scenario in which you have shown up to a datacenter with only a laptop and your task is to transition a rack of bare metal machines into an HA management plane and multiple Kubernetes clusters created by that management plane. In this guide, we will go through how to create a bootstrap cluster using a Docker-based Talos cluster, provision the management plane, and pivot over to it. Guides around post-pivoting setup and subsequent cluster creation should also be found in the “Guides” section of the sidebar.

Because of the design of Cluster API, there is inherently a “chicken and egg” problem with needing a Kubernetes cluster in order to provision the management plane. Talos Systems and the Cluster API community have created tools to help make this transition easier.

Prerequisites

First, you need to install the latest `talosctl` by running the following script:

```
curl -Lo /usr/local/bin/talosctl https://github.com/talos-systems/talos/releases/latest/download/talosctl-$(uname -s |
chmod +x /usr/local/bin/talosctl
```

You can read more about Talos and `talosctl` at talos.dev.

Next, there are two big prerequisites involved with bootstrapping Sidero: routing and DHCP setup.

From the routing side, the laptop from which you are bootstrapping *must* be accessible by the bare metal machines that we will be booting. In the datacenter scenario described above, the easiest way to achieve this is probably to hook the laptop onto the server rack’s subnet by plugging it into the top-of-rack switch. This is needed for TFTP, PXE booting, and for the ability to register machines with the bootstrap plane.

DHCP configuration is needed to tell the metal servers what their “next server” is when PXE booting. The configuration of this is different for each environment and each DHCP server, thus it’s impossible to give an easy guide. However, here is an example of the configuration for an Ubiquti EdgeRouter that uses vyatta-dhcpd as the DHCP service:

This block shows the subnet setup, as well as the extra “subnet-parameters” that tell the DHCP server to include the ipxe-metal.conf file.

> These commands are run under the `configure` option in EdgeRouter

```
$ show service dhcp-server shared-network-name MetalDHCP

 authoritative enable
 subnet 192.168.254.0/24 {
     default-router 192.168.254.1
     dns-server 192.168.1.200
     lease 86400
     start 192.168.254.2 {
         stop 192.168.254.252
     }
     subnet-parameters "include &quot;/config/ipxe-metal.conf&quot;;"
 }
```

Here is the `ipxe-metal.conf` file.

```
$ cat /config/ipxe-metal.conf

allow bootp;
allow booting;

next-server 192.168.1.150;
filename "snp.efi"; # use "undionly.kpxe" for BIOS netboot or "snp.efi" for UEFI netboot

host talos-mgmt-0 {
    fixed-address 192.168.254.2;
    hardware ethernet d0:50:99:d3:33:60;
}
```

If you want to boot multiple architectures, you can use the *DHCP Option 93* to specify the architecture.

First we need to define *option 93* in the DHCP server configuration.

```
set service dhcp-server global-parameters "option system-arch code 93 = unsigned integer 16;"
```

Now we can specify condition based on *option 93* in `ipxe-metal.conf` file

```
$ cat /config/ipxe-metal.conf

allow bootp;
allow booting;

next-server 192.168.1.150;

if option system-arch = 00:0b {
    filename "snp-arm64.efi";
} else {
    filename "snp.efi";
}

host talos-mgmt-0 {
    fixed-address 192.168.254.2;
    hardware ethernet d0:50:99:d3:33:60;
}
```

Notice that it sets a static address for the management node that I'll be booting, in addition to providing the "next server" info. This "next server" IP address will match references to `PUBLIC_IP` found below in this guide.

Create a Local Cluster

The `talosctl` CLI tool has built-in support for spinning up Talos in docker containers. Let's use this to our advantage as an easy Kubernetes cluster to start from.

Set an environment variable called `PUBLIC_IP` which is the "public" IP of your machine. Note that "public" is a bit of a misnomer. We're really looking for the IP of your machine, not the IP of the node on the docker bridge (ex: `192.168.1.150`).

```
export PUBLIC_IP="192.168.1.150"
```

We can now create our Docker cluster. Issue the following to create a single-node cluster:

```
talosctl cluster create \
 --kubernetes-version 1.29.0 \
  -p 69:69/udp,8081:8081/tcp,51821:51821/udp \
  --workers 0 \
  --endpoint $PUBLIC_IP
```

Note that there are several ports mentioned in the command above. These allow us to access the services that will get deployed on this node.

Once the cluster create command is complete, issue `talosctl kubeconfig /desired/path` to fetch the kubeconfig for this cluster. You should then set your `KUBECONFIG` environment variable to the path of this file.

Untaint Control Plane

Because this is a single node cluster, we need to remove the "NoSchedule" taint on the node to make sure non-controlplane components can be scheduled.

```
kubectl taint node talos-default-controlplane-1 node-role.kubernetes.io/control-plane:NoSchedule-
```

Install Sidero

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

```
SIDERO_CONTROLLER_MANAGER_HOST_NETWORK=true \
  SIDERO_CONTROLLER_MANAGER_DEPLOYMENT_STRATEGY=Recreate \
  SIDERO_CONTROLLER_MANAGER_API_ENDPOINT=$PUBLIC_IP \
  clusterctl init -b talos -c talos -i sidero
```

We will now want to ensure that the Sidero services that got created are publicly accessible across our subnet. These variables above will allow the metal machines to speak to these services later.

Register the Servers

At this point, any servers on the same network as Sidero should PXE boot using the Sidero PXE service. To register a server with Sidero, simply turn it on and Sidero will do the rest. Once the registration is complete, you should see the servers registered with `kubectl get servers`:

```
$ kubectl get servers -o wide
NAME                                   HOSTNAME        ACCEPTED    ALLOCATED    CLEAN
00000000-0000-0000-0000-d05099d33360   192.168.254.2   false       false        false
```

Setting up 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. See IPMI for more information.

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

```
kubectl patch server 00000000-0000-0000-0000-d05099d33360 --type='json' -p='[{"op": "add", "path": "/spec/bmc", "value'
```

If IPMI info is not set, servers should be configured to boot first from network, then from disk.

Configuring the installation disk

Note that for bare-metal setup, you would need to specify an installation disk. See Installation Disk for details on how to do this. You should configure this before accepting the server.

Accept the Servers

Note in the output above that the newly registered servers are not `accepted`. In order for a server to be eligible for consideration, it *must* be marked as `accepted`. Before a `Server` is accepted, no write action will be performed against it. Servers can be accepted by issuing a patch command like:

```
kubectl patch server 00000000-0000-0000-0000-d05099d33360 --type='json' -p='[{"op": "replace", "path": "/spec/accepted'
```

For more information on server acceptance, see the server docs.

Create Management Plane

We are now ready to template out our management plane. Using clusterctl, we can create a cluster manifest with:

```
clusterctl generate cluster management-plane -i sidero > management-plane.yaml
```

Note that there are several variables that should be set in order for the templating to work properly:

- `CONTROL_PLANE_ENDPOINT` and `CONTROL_PLANE_PORT`: The endpoint (IP address or hostname) and the port used for the Kubernetes API server (e.g. for `https://1.2.3.4:6443`: `CONTROL_PLANE_ENDPOINT=1.2.3.4` and `CONTROL_PLANE_PORT=6443`). This is the equivalent of the `endpoint` you would specify in `talosctl gen config`. There are a variety of ways to configure a control plane endpoint. Some common ways for an HA setup are to use DNS, a load balancer, or BGP. A simpler method is to use the IP of a single node. This has the disadvantage of being a single point of failure, but it can be a simple way to get running.
- `CONTROL_PLANE_SERVERCLASS`: The server class to use for control plane nodes.
- `WORKER_SERVERCLASS`: The server class to use for worker nodes.
- `KUBERNETES_VERSION`: The version of Kubernetes to deploy (e.g. `v1.29.0`).
- `CONTROL_PLANE_PORT`: The port used for the Kubernetes API server (port 6443)
- `TALOS_VERSION`: This should correspond to the minor version of Talos that you will be deploying (e.g. `v1.6.1`). This value is used in determining the fields present in the machine configuration that gets generated for Talos nodes.

For instance:

```
export CONTROL_PLANE_SERVERCLASS=any
export WORKER_SERVERCLASS=any
export TALOS_VERSION=v1.6.1
export KUBERNETES_VERSION=v1.29.0
export CONTROL_PLANE_PORT=6443
export CONTROL_PLANE_ENDPOINT=1.2.3.4
clusterctl generate cluster management-plane -i sidero > management-plane.yaml
```

In addition, you can specify the replicas for control-plane & worker nodes in management-plane.yaml manifest for TalosControlPlane and MachineDeployment objects. Also, they can be scaled if needed (after applying the `management-plane.yaml` manifest):

```
kubectl get taloscontrolplane
kubectl get machinedeployment
kubectl scale taloscontrolplane management-plane-cp --replicas=3
```

Now that we have the manifest, we can simply apply it:

```
kubectl apply -f management-plane.yaml
```

NOTE: The templated manifest above is meant to act as a starting point. If customizations are needed to ensure proper setup of your Talos cluster, they should be added before applying.

Once the management plane is setup, you can fetch the talosconfig by using the cluster label. Be sure to update the cluster name and issue the following command:

```
kubectl get talosconfig \
  -l cluster.x-k8s.io/cluster-name=<CLUSTER NAME> \
  -o yaml -o jsonpath='{.items[0].status.talosConfig}' > management-plane-talosconfig.yaml
```

With the talosconfig in hand, the management plane's kubeconfig can be fetched with `talosctl --talosconfig management-plane-talosconfig.yaml kubeconfig`

Pivoting

Once we have the kubeconfig for the management cluster, we now have the ability to pivot the cluster from our bootstrap. Using clusterctl, issue:

```
clusterctl init --kubeconfig=/path/to/management-plane/kubeconfig -i sidero -b talos -c talos
```

Followed by:

```
clusterctl move --to-kubeconfig=/path/to/management-plane/kubeconfig
```

Upon completion of this command, we can now tear down our bootstrap cluster with `talosctl cluster destroy` and begin using our management plane as our point of creation for all future clusters!.

2 - Building A Management Plane with ISO Image

A guide for bootstrapping Sidero management plane using the ISO image

This guide will provide some very basic detail about how you can also build a Sidero management plane using the Talos ISO image instead of following the Docker-based process that we detail in our Getting Started tutorials.

Using the ISO is a perfectly valid way to build a Talos cluster, but this approach is not recommended for Sidero as it avoids the "pivot" step detailed here. Skipping this step means that the management plane does not become "self-hosted", in that it cannot be upgraded and scaled using the Sidero processes we follow for workload clusters. For folks who are willing to take care of their management plane in other ways, however, this approach will work fine.

The rough outline of this process is very short and sweet, as it relies on other documentation:

- For each management plane node, boot the ISO and install Talos using the "apply-config" process mentioned in our Talos Getting Started docs. These docs go into heavy detail on using the ISO, so they will not be recreated here.
- With a Kubernetes cluster now in hand (and with access to it via `talosctl` and `kubectl`), you can simply pickup the Getting Started tutorial at the "Install Sidero" section here. Keep in mind, however, that you will be unable to do the "pivoting" section of the tutorial, so just skip that step when you reach the end of the tutorial.

Note: It may also be of interest to view the prerequisite guides on CLI and DHCP setup, as they will still apply to this method.

- For long-term maintenance of a management plane created in this way, refer to the Talos documentation for upgrading Kubernetes and Talos itself.

3 - Decommissioning Servers

A guide for decommissioning servers

This guide will detail the process for removing a server from Sidero. The process is fairly simple with a few pieces of information.

- For the given server, take note of any serverclasses that are configured to match the server.
- Take note of any clusters that make use of aforementioned serverclasses.
- For each matching cluster, edit the cluster resource with `kubectl edit cluster` and set `.spec.paused` to `true` . Doing this ensures that no new machines will get created for these servers during the decommissioning process.
- If you want to mark a server to be not allocated after it's accepted into the cluster, set the `.spec.cordoned` field to `true` . This will prevent the server from being allocated to any new clusters (still allowing it to be wiped).
- If the server is already part of a cluster (`kubectl get serverbindings -o wide` should provide this info), you can now delete the machine that corresponds with this server via `kubectl delete machine <machine_name>` .
- With the machine deleted, Sidero will reboot the machine and wipe its disks.
- Once the disk wiping is complete and the server is turned off, you can finally delete the server from Sidero with `kubectl delete server <server_name>` and repurpose the server for something else.
- Finally, unpause any clusters that were edited in step 3 by setting `.spec.paused` to `false` .

4 - Creating Your First Cluster

A guide for creating your first cluster with the Sidero management plane

Introduction

This guide will detail the steps needed to provision your first bare metal Talos cluster after completing the bootstrap and pivot steps detailed in the previous guide. There will be two main steps in this guide: reconfiguring the Sidero components now that they have been pivoted and the actual cluster creation.

Reconfigure Sidero

Patch Services

In this guide, we will convert the services to use host networking. This is also necessary because some protocols like TFTP don't allow for port configuration. Along with some nodeSelectors and a scale up of the metal controller manager deployment, creating the services this way allows for the creation of DNS names that point to all management plane nodes and provide an HA experience if desired. It should also be noted, however, that there are many options for achieving this functionality. Users can look into projects like MetalLB or KubeRouter with BGP and ECMP if they desire something else.

Metal Controller Manager:

```
## Use host networking
kubectl patch deploy -n sidero-system sidero-controller-manager --type='json' -p='[{"op": "add", "path": "/spec/templat
```

Update Environment

Sidero by default appends `talos.config` kernel argument with based on the flags `--api-endpoint` and `--api-port` to the `sidero-controller-manager` : `talos.config=http://$API_ENDPOINT:$API_PORT/configdata?uuid=` .

If this default value doesn't apply, edit the environment with `kubectl edit environment default` and add the `talos.config` kernel arg with the IP of one of the management plane nodes (or the DNS entry you created).

Update DHCP

The DHCP options configured in the previous guide should now be updated to point to your new management plane IP or to the DNS name if it was created.

A revised ipxe-metal.conf file looks like:

```
allow bootp;
allow booting;

next-server 192.168.254.2;
if exists user-class and option user-class = "iPXE" {
  filename "http://192.168.254.2:8081/boot.ipxe";
} else {
  if substring (option vendor-class-identifier, 15, 5) = "00000" {
    # BIOS
    if substring (option vendor-class-identifier, 0, 10) = "HTTPClient" {
      option vendor-class-identifier "HTTPClient";
      filename "http://192.168.254.2:8081/tftp/undionly.kpxe";
    } else {
      filename "undionly.kpxe";
    }
  } else {
    # UEFI
    if substring (option vendor-class-identifier, 0, 10) = "HTTPClient" {
      option vendor-class-identifier "HTTPClient";
      filename "http://192.168.254.2:8081/tftp/snp.efi";
    } else {
      filename "snp.efi";
    }
  }
}

host talos-mgmt-0 {
   fixed-address 192.168.254.2;
   hardware ethernet d0:50:99:d3:33:60;
}
```

There are multiple ways to boot the via iPXE:

- if the node has built-in iPXE, direct URL to the iPXE script can be used: `http://192.168.254.2:8081/boot.ipxe` .
- depending on the boot mode (BIOS or UEFI), either `snp.efi` or `undionly.kpxe` can be used (these images contain embedded iPXE scripts).
- iPXE binaries can be delivered either over TFTP or HTTP (HTTP support depends on node firmware).

Register the Servers

At this point, any servers on the same network as Sidero should PXE boot using the Sidero PXE service. To register a server with Sidero, simply turn it on and Sidero will do the rest. Once the registration is complete, you should see the servers registered with `kubectl get servers` :

```
$ kubectl get servers -o wide
NAME                                   HOSTNAME        ACCEPTED   ALLOCATED   CLEAN
00000000-0000-0000-0000-d05099d33360   192.168.254.2   false      false       false
```

Accept the Servers

Note in the output above that the newly registered servers are not `accepted` . In order for a server to be eligible for consideration, it *must* be marked as `accepted` . Before a `Server` is accepted, no write action will be performed against it. Servers can be accepted by issuing a patch command like:

```
kubectl patch server 00000000-0000-0000-0000-d05099d33360 --type='json' -p='[{"op": "replace", "path": "/spec/accepted'
```

For more information on server acceptance, see the server docs.

Create the Cluster

The cluster creation process should be identical to what was detailed in the previous guide. Using clusterctl, we can create a cluster manifest with:

```
clusterctl generate cluster workload-cluster -i sidero > workload-cluster.yaml
```

Note that there are several variables that should be set in order for the templating to work properly:

- `CONTROL_PLANE_ENDPOINT` and `CONTROL_PLANE_PORT` : The endpoint (IP address or hostname) and the port used for the Kubernetes API server (e.g. for `https://1.2.3.4:6443` : `CONTROL_PLANE_ENDPOINT=1.2.3.4` and `CONTROL_PLANE_PORT=6443`). This is the equivalent of the `endpoint` you would specify in `talosctl gen config` . There are a variety of ways to configure a control plane endpoint. Some common ways for an HA setup are to use DNS, a load balancer, or BGP. A simpler method is to use the IP of a single node. This has the disadvantage of being a single point of failure, but it can be a simple way to get running.
- `CONTROL_PLANE_SERVERCLASS` : The server class to use for control plane nodes.
- `WORKER_SERVERCLASS` : The server class to use for worker nodes.
- `KUBERNETES_VERSION` : The version of Kubernetes to deploy (e.g. `v1.19.4`).
- `TALOS_VERSION` : This should correspond to the minor version of Talos that you will be deploying (e.g. `v0.10`). This value is used in determining the fields present in the machine configuration that gets generated for Talos nodes. Note that the default is currently `v0.13` .

Now that we have the manifest, we can simply apply it:

```
kubectl apply -f workload-cluster.yaml
```

NOTE: The templated manifest above is meant to act as a starting point. If customizations are needed to ensure proper setup of your Talos cluster, they should be added before applying.

Once the workload cluster is setup, you can fetch the talosconfig with a command like:

```
kubectl get talosconfig -o yaml workload-cluster-cp-xxx -o jsonpath='{.status.talosConfig}' > workload-cluster-taloscor
```

Then the workload cluster’s kubeconfig can be fetched with `talosctl --talosconfig workload-cluster-talosconfig.yaml kubeconfig /desired/path` .

5 - Patching

A guide describing patching

Server resources can be updated by using the `configPatches` section of the custom resource. Any field of the Talos machine config can be overridden on a per-machine basis using this method. The format of these patches is based on JSON 6902 that you may be used to in tools like kustomize.

Any patches specified in the server resource are processed by the Sidero controller before it returns a Talos machine config for a given server at boot time.

A set of patches may look like this:

```yaml
apiVersion: metal.sidero.dev/v1alpha2
kind: Server
metadata:
  name: 00000000-0000-0000-0000-d05099d33360
spec:
  configPatches:
    - op: replace
      path: /machine/install
      value:
        disk: /dev/sda
    - op: replace
      path: /cluster/network/cni
      value:
        name: "custom"
        urls:
          - "http://192.168.1.199/assets/cilium.yaml"
```

Testing Configuration Patches

While developing config patches it is usually convenient to test generated config with patches before actual server is provisioned with the config.

This can be achieved by querying the metadata server endpoint directly:

```
$ curl http://$PUBLIC_IP:8081/configdata?uuid=$SERVER_UUID
version: v1alpha1
...
```

Replace `$PUBLIC_IP` with the Sidero IP address and `$SERVER_UUID` with the name of the `Server` to test against.

If metadata endpoint returns an error on applying JSON patches, make sure config subtree being patched exists in the config. If it doesn't exist, create it with the `op: add` above the `op: replace` patch.

Combining Patches from Multiple Sources

Config patches might be combined from multiple sources (`Server` , `ServerClass` , `TalosControlPlane` , `TalosConfigTemplate`), which is explained in details in Metadata section.

6 - Provisioning Flow

Diagrams for various flows in Sidero.

```
graph TD;
    Start(Start);
    End(End);

    %% Decisions

    IsOn{Is server is powered on?};
    IsRegistered{Is server is registered?};
    IsAccepted{Is server is accepted?};
    IsClean{Is server is clean?};
    IsAllocated{Is server is allocated?};

    %% Actions

    DoPowerOn[Power server on];
    DoPowerOff[Power server off];
    DoBootAgentEnvironment[Boot agent];
    DoBootEnvironment[Boot environment];
    DoRegister[Register server];
    DoWipe[Wipe server];

    %% Chart

    Start-->IsOn;
    IsOn--Yes-->End;
    IsOn--No-->DoPowerOn;

    DoPowerOn--->IsRegistered;

    IsRegistered--Yes--->IsAccepted;
    IsRegistered--No--->DoBootAgentEnvironment-->DoRegister;

    DoRegister-->IsRegistered;

    IsAccepted--Yes--->IsAllocated;
    IsAccepted--No--->End;

    IsAllocated--Yes--->DoBootEnvironment;
    IsAllocated--No--->IsClean;
    IsClean--No--->DoWipe-->DoPowerOff;

    IsClean--Yes--->DoPowerOff;

    DoBootEnvironment-->End;

    DoPowerOff-->End;
```

Installation Flow

```
graph TD;
    Start(Start);
    End(End);

    %% Decisions

    IsInstalled{Is installed};

    %% Actions

    DoInstall[Install];
    DoReboot[Reboot];

    %% Chart

    Start-->IsInstalled;
    IsInstalled--Yes-->End;
    IsInstalled--No-->DoInstall;

    DoInstall-->DoReboot;

    DoReboot-->IsInstalled;
```

7 - Raspberry Pi4 as Servers

Using Raspberrypi Pi 4 as servers

This guide will explain on how to use Sidero to manage Raspberrypi-4's as servers. This guide goes hand in hand with the bootstrapping guide.

From the bootstrapping guide, reach "Install Sidero" and come back to this guide. Once you finish with this guide, you will need to go back to the bootstrapping guide and continue with "Register the servers".

The rest of this guide goes with the assumption that you've a cluster setup with Sidero and ready to accept servers. This guide will explain the changes that needs to be made to be able to accept RPI4 as server.

RPI4 boot process

To be able to boot talos on the Pi4 via network, we need to undergo a 2-step boot process. The Pi4 has an EEPROM which contains code to boot up the Pi. This EEPROM expects a specific boot folder structure as explained on this page. We will use the EEPROM to boot into UEFI, which we will then use to PXE and iPXE boot into sidero & talos.

Prerequisites

Update EEPROM

NOTE: If you've updated the EEPROM with the image that was referenced on the talos docs, you can either flash it with the one mentioned below, or visit the EEPROM config docs and change the boot order of EEPROM to `0xf21`. Which means try booting from SD first, then try network.

To enable the EEPROM on the Pi to support network booting, we must update it to the latest version. Visit the release page and grab the latest `rpi-boot-eeprom-recovery-*-network.zip` (as of time of writing, v2021.0v.29-138a1 was used). Put this on a SD card and plug it into the Pi. The Pi's status light will flash rapidly after a few seconds, this indicates that the EEPROM has been updated.

This operation needs to be done once per Pi.

Serial number

Power on the Pi without an SD card in it and hook it up to a monitor, you will be greeted with the boot screen. On this screen you will find some information about the Pi. For this guide, we are only interested in the serial number. The first line under the Pi logo will be something like the following:

```
board: xxxxxx <serial> <MAC address>
```

Write down the 8 character serial.

talos-systems/pkg

Clone the talos-systems/pkg repo. Create a new folder called `raspberrypi4-uefi` and `raspberrypi4-uefi/serials`. Create a file `raspberrypi4-uefi/pkg.yaml` containing the following:

```
name: raspberrypi4-uefi
variant: alpine
install:
  - unzip
steps:
# {{ if eq .ARCH "aarch64" }} This in fact is YAML comment, but Go templating instruction is evaluated by bldr restrict
  - sources:
      - url: https://github.com/pftf/RPi4/releases/download/v1.26/RPi4_UEFI_Firmware_v1.26.zip # <-- update version NR
        destination: RPi4_UEFI_Firmware.zip
        sha256: d6db87484dd98dfbeb64eef203944623130cec8cb71e553eab21f8917e0285f7
        sha512: 96a71086cdd062b51ef94726ebcbf15482b70c56262555a915499bafc04aff959d122410af37214760eda8534b58232a64f6a8a
    prepare:
      - |
        unzip RPi4_UEFI_Firmware.zip
        rm RPi4_UEFI_Firmware.zip
        mkdir /rpi4
        mv ./* /rpi4
    install:
      - |
        mkdir /tftp
        ls /pkg/serials | while read serial; do mkdir /tftp/$serial && cp -r /rpi4/* /tftp/$serial && cp -r /pkg/seria
# {{ else }}
  - install:
      - |
                mkdir -p /tftp
# {{ end }}
finalize:
  - from: /
    to: /
```

UEFI / RPi4

Now that the EEPROM can network boot, we need to prepare the structure of our boot folder. Essentially what the bootloader will do is look for this folder on the network rather than on the SD card.

Visit the release page of RPi4 and grab the latest `RPi4_UEFI_Firmware_v*.zip` (at the time of writing, v1.26 was used). Extract the zip into a folder, the structure will look like the following:

```
.
├── RPI_EFI.fd
├── RPi4_UEFI_Firmware_v1.26.zip
├── Readme.md
├── bcm2711-rpi-4-b.dtb
├── bcm2711-rpi-400.dtb
├── bcm2711-rpi-cm4.dtb
├── config.txt
├── firmware
│   ├── LICENCE.txt
│   ├── Readme.txt
│   ├── brcmfmac43455-sdio.bin
│   ├── brcmfmac43455-sdio.clm_blob
│   └── brcmfmac43455-sdio.txt
├── fixup4.dat
├── overlays
│   └── miniuart-bt.dtbo
└── start4.elf
```

As a one time operation, we need to configure UEFI to do network booting by default, remove the 3gb mem limit if it's set and optionally set the CPU clock to max. Take these files and put them on the SD card and boot the Pi. You will see the Pi logo, and the option to hit `esc`.

Remove 3GB mem limit

1. From the home page, visit "Device Manager".

2. Go down to "Raspberry Pi Configuration" and open that menu.
3. Go to "Advanced Configuration".
4. Make sure the option "Limit RAM to 3 GB" is set to `Disabled` .

Change CPU to Max (optionally)

1. From the home page, visit "Device Manager".
2. Go down to "Raspberry Pi Configuration" and open that menu.
3. Go to "CPU Configuration".
4. Change CPU clock to `Max` .

Change boot order

1. From the home page, visit "Boot Maintenance Manager".
2. Go to "Boot Options".
3. Go to "Change Boot Order".
4. Make sure that `UEFI PXEv4` is the first boot option.

Persisting changes

Now that we have made the changes above, we need to persist these changes. Go back to the home screen and hit `reset` to save the changes to disk.

When you hit `reset` , the settings will be saved to the `RPI_EFI.fd` file on the SD card. This is where we will run into a limitation that is explained in the following issue: pftf/RPi4#59. What this mean is that we need to create a `RPI_EFI.fd` file for each Pi that we want to use as server. This is because the MAC address is also stored in the `RPI_EFI.fd` file, which makes it invalid when you try to use it in a different Pi.

Plug the SD card back into your computer and extract the `RPI_EFI.fd` file from it and place it into the `raspberrypi4-uefi/serials/<serial>/` . The dir should look like this:

```
raspberrypi4-uefi/
├── pkg.yaml
└── serials
    └── XXXXXXXX
        └── RPI_EFI.fd
```

Build the image with the boot folder contents

Now that we have the `RPI_EFI.fd` of our Pi in the correct location, we must now build a docker image containing the boot folder for the EEPROM. To do this, run the following command in the pkgs repo:

`make PLATFORM=linux/arm64 USERNAME=$USERNAME PUSH=true TARGETS=raspberrypi4-uefi`

This will build and push the following image: `ghcr.io/$USERNAME/raspberrypi4-uefi:<tag>`

If you need to change some other settings like registry etc, have a look in the Makefile to see the available variables that you can override.

The content of the `/tftp` folder in the image will be the following:

```
XXXXXXXX
├── RPI_EFI.fd
├── Readme.md
├── bcm2711-rpi-4-b.dtb
├── bcm2711-rpi-400.dtb
├── bcm2711-rpi-cm4.dtb
├── config.txt
├── firmware
│   ├── LICENCE.txt
│   ├── Readme.txt
│   ├── brcmfmac43455-sdio.bin
│   ├── brcmfmac43455-sdio.clm_blob
│   └── brcmfmac43455-sdio.txt
├── fixup4.dat
├── overlays
│   └── miniuart-bt.dtbo
└── start4.elf
```

Patch metal controller

To enable the 2 boot process, we need to include this EEPROM boot folder into the sidero's tftp folder. To achieve this, we will use an init container using the image we created above to copy the contents of it into the tftp folder.

Create a file `patch.yaml` with the following contents:

```yaml
spec:
  template:
    spec:
      volumes:
        - name: tftp-folder
          emptyDir: {}
      initContainers:
      - image: ghcr.io/<USER>/raspberrypi4-uefi:v<TAG> # <-- change accordingly.
        imagePullPolicy: Always
        name: tftp-folder-setup
        command:
          - cp
        args:
          - -r
          - /tftp
          - /var/lib/sidero/
        volumeMounts:
          - mountPath: /var/lib/sidero/tftp
            name: tftp-folder
      containers:
      - name: manager
        volumeMounts:
          - mountPath: /var/lib/sidero/tftp
            name: tftp-folder
```

Followed by this command to apply the patch:

```
kubectl -n sidero-system patch deployments.apps sidero-controller-manager --patch "$(cat patch.yaml)"
```

Configure BootFromDiskMethod

By default, Sidero will use iPXE's `exit` command to attempt to force boot from disk. On Raspberry Pi, this will drop you into the bootloader interface, and you will need to connect a keyboard and manually select the disk to boot from.

The BootFromDiskMethod can be configured on individual Servers, on ServerClasses, or as a command-line argument to the Sidero metal controller itself (`--boot-from-disk-method=<value>`). In order to force the Pi to use the configured bootloader order, the BootFromDiskMethod needs to be set to `ipxe-sanboot` .

Profit

With the patched metal controller, you should now be able to register the Pi4 to sidero by just connecting it to the network. From this point you can continue with the bootstrapping guide.

8 - Sidero on Raspberry Pi 4

Running Sidero on Raspberry Pi 4 to provision bare-metal servers.

Sidero doesn't require a lot of computing resources, so SBCs are a perfect fit to run the Sidero management cluster. In this guide, we are going to install Talos on Raspberry Pi4, deploy Sidero and other CAPI components.

Prerequisites

Please see Talos documentation for additional information on installing Talos on Raspberry Pi4.

Download the `clusterctl` CLI from CAPI releases. The minimum required version is 1.5.0.

Installing Talos

Prepare the SD card with the Talos RPi4 image, and boot the RPi4. Talos should drop into maintenance mode printing the acquired IP address. Record the IP address as the environment variable `SIDERO_ENDPOINT` :

```
export SIDERO_ENDPOINT=192.168.x.x
```

Note: it makes sense to transform DHCP lease for RPi4 into a static reservation so that RPi4 always has the same IP address.

Generate Talos machine configuration for a single-node cluster:

```
talosctl gen config --config-patch='[{"op": "add", "path": "/cluster/allowSchedulingOnControlPlanes", "value": true},{'
```

Submit the generated configuration to Talos:

```
talosctl apply-config --insecure -n ${SIDERO_ENDPOINT} -f controlplane.yaml
```

Merge client configuration `talosconfig` into default `~/.talos/config` location:

```
talosctl config merge talosconfig
```

Update default endpoint and nodes:

```
talosctl config endpoints ${SIDERO_ENDPOINT}
talosctl config nodes ${SIDERO_ENDPOINT}
```

You can verify that Talos has booted by running:

```
$ talosctl version
talosctl version
Client:
    Tag:         v0.10.3
    SHA:         21018f28
    Built:
    Go version:  go1.16.3
    OS/Arch:     linux/amd64

Server:
    NODE:        192.168.0.31
    Tag:         v0.10.3
    SHA:         8f90c6a8
    Built:
    Go version:  go1.16.3
    OS/Arch:     linux/arm64
```

Bootstrap the etcd cluster:

```
talosctl bootstrap
```

At this point, Kubernetes is bootstrapping, and it should be available once all the images are fetched.

Fetch the `kubeconfig` from the cluster with:

```
talosctl kubeconfig
```

You can watch the bootstrap progress by running:

```
talosctl dmesg -f
```

Once Talos prints `[talos] boot sequence: done` , Kubernetes should be up:

```
kubectl get nodes
```

Installing Sidero

Install Sidero with host network mode, exposing the endpoints on the node's address:

```
SIDERO_CONTROLLER_MANAGER_HOST_NETWORK=true SIDERO_CONTROLLER_MANAGER_DEPLOYMENT_STRATEGY=Recreate SIDERO_CONTROLLER_MA
```

Watch the progress of installation with:

```
watch -n 2 kubectl get pods -A
```

Once images are downloaded, all pods should be in running state:

```
$ kubectl get pods -A
NAMESPACE             NAME                                          READY   STATUS    RESTARTS   AGE
cabpt-system          cabpt-controller-manager-6458494888-d7lnm     1/1     Running   0          29m
cacppt-system         cacppt-controller-manager-f98854db8-qgkf9     1/1     Running   0          29m
capi-system           capi-controller-manager-58f797cb65-8dwpz      2/2     Running   0          30m
capi-webhook-system   cabpt-controller-manager-85fd964c9c-ldzb6     1/1     Running   0          29m
capi-webhook-system   cacppt-controller-manager-75c479b7f-5hw89     1/1     Running   0          29m
capi-webhook-system   capi-controller-manager-7d596cc4cb-kjrfk      2/2     Running   0          30m
capi-webhook-system   caps-controller-manager-79664cf677-zqbvw      1/1     Running   0          29m
cert-manager          cert-manager-86cb5dcfdd-v86wr                 1/1     Running   0          31m
cert-manager          cert-manager-cainjector-84cf775b89-swk25      1/1     Running   0          31m
cert-manager          cert-manager-webhook-7f9f4f8dcb-29xm4         1/1     Running   0          31m
kube-system           coredns-fcc4c97fb-wkxkg                       1/1     Running   0          35m
kube-system           coredns-fcc4c97fb-xzqzj                       1/1     Running   0          35m
kube-system           kube-apiserver-talos-192-168-0-31             1/1     Running   0          33m
kube-system           kube-controller-manager-talos-192-168-0-31    1/1     Running   0          33m
kube-system           kube-flannel-qmlw6                            1/1     Running   0          34m
kube-system           kube-proxy-j24hg                              1/1     Running   0          34m
kube-system           kube-scheduler-talos-192-168-0-31             1/1     Running   0          33m
```

Verify Sidero installation and network setup with:

```
$ curl -I http://${SIDERO_ENDPOINT}:8081/tftp/ipxe.efi
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 1020416
Content-Type: application/octet-stream
Last-Modified: Thu, 03 Jun 2021 15:40:58 GMT
Date: Thu, 03 Jun 2021 15:41:51 GMT
```

Now Sidero is installed, and it is ready to be used. Configure your DHCP server to PXE boot your bare metal servers from `$SIDERO_ENDPOINT` (see Bootstrapping guide on DHCP configuration).

Backup and Recovery

SD cards are not very reliable, so make sure you are taking regular etcd backups, so that you can recover your Sidero installation in case of data loss.