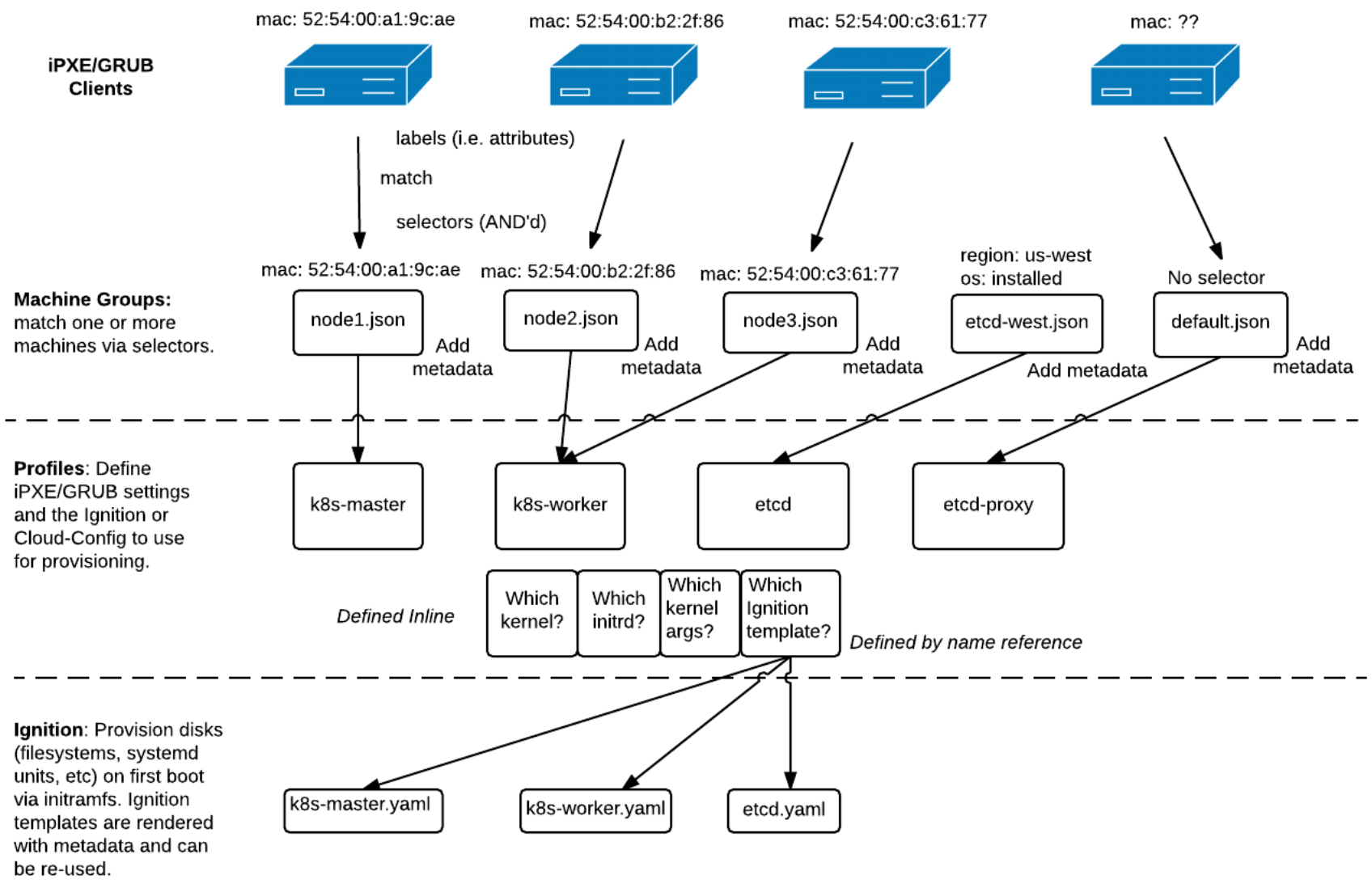
matchbox

`matchbox` is an HTTP and gRPC service that renders signed Ignition configs, cloud-configs, network boot configs, and metadata to machines to create CoreOS Container Linux clusters. `matchbox` maintains **Group** definitions which match machines to *profiles* based on labels (e.g. MAC address, UUID, stage, region). A **Profile** is a named set of config templates (e.g. iPXE, GRUB, Ignition config, Cloud-Config, generic configs). The aim is to use Container Linux's early-boot capabilities to provision Container Linux machines.

Network boot endpoints provide PXE, iPXE, GRUB support. `matchbox` can be deployed as a binary, as an appc container with rkt, or as a Docker container.



Getting started

Get started running `matchbox` on your Linux machine, with rkt or Docker.

- matchbox with rkt
- matchbox with Docker

Flags

See configuration flags and variables.

API

- HTTP API
- gRPC API

Data

A `Store` stores machine Groups, Profiles, and associated Ignition configs, cloud-configs, and generic configs. By default, `matchbox` uses a `FileStore` to search a `-data-path` for these resources.

Prepare `/var/lib/matchbox` with `groups`, `profile`, `ignition`, `cloud`, and `generic` subdirectories. You may wish to keep these files under version control.

```
/var/lib/matchbox
├── cloud
│   ├── cloud.yaml.tmpl
│   └── worker.sh.tmpl
├── ignition
│   └── raw.ign
│   └── etcd.yaml.tmpl
│   └── simple.yaml.tmpl
├── generic
│   └── config.yaml
│   └── setup.cfg
│   └── datacenter-1.tmpl
├── groups
│   └── default.json
│   └── node1.json
│   └── us-central1-a.json
└── profiles
    └── etcd.json
    └── worker.json
```

The examples directory is a valid data directory with some pre-defined configs. Note that `examples/groups` contains many possible groups in nested directories for demo purposes (tutorials pick one to mount). Your machine groups should be kept directly inside the `groups` directory as shown above.

Profiles

Profiles reference an Ignition config, Cloud-Config, and/or generic config by name and define network boot settings.

```
{
  "id": "etcd",
  "name": "Container Linux with etcd2",
  "cloud_id": "",
  "ignition_id": "etcd.yaml",
  "generic_id": "some-service.cfg",
  "boot": {
    "kernel": "/assets/coreos/1911.4.0/coreos_production_pxe.vmlinuz",
    "initrd": ["/assets/coreos/1911.4.0/coreos_production_pxe_image.cpio.gz"],
    "args": [
      "coreos.config.url=http://matchbox.foo:8080/ignition?uuid=${uuid}&mac=${mac:hexhyp}",
      "coreos.first_boot=yes",
      "coreos.autologin"
    ]
  },
}
```

The `"boot"` settings will be used to render configs to network boot programs such as iPXE or GRUB. You may reference remote kernel and initrd assets or local assets.

To use Ignition, set the `coreos.config.url` kernel option to reference the `matchbox` Ignition endpoint, which will render the `ignition_id` file. Be sure to add the `coreos.first_boot` option as well.

To use cloud-config, set the `cloud-config-url` kernel option to reference the `matchbox` Cloud-Config endpoint, which will render the `cloud_id` file.

Groups

Groups define selectors which match zero or more machines. Machine(s) matching a group will boot and provision according to the group's `Profile`.

Create a group definition with a `Profile` to be applied, selectors for matching machines, and any `metadata` needed to render templated configs. For example `/var/lib/matchbox/groups/node1.json` matches a single machine with MAC address `52:54:00:89:d8:10`.

```
# /var/lib/matchbox/groups/node1.json
{
  "name": "node1",
  "profile": "etcd",
  "selector": {
    "mac": "52:54:00:89:d8:10"
  },
  "metadata": {
    "fleet_metadata": "role=etcd,name=node1",
    "etcd_name": "node1",
    "etcd_initial_cluster": "node1=http://node1.example.com:2380,node2=http://node2.example.com:2380,node3=http://noc
  }
}
```

Meanwhile, `/var/lib/matchbox/groups/proxy.json` acts as the default machine group since it has no selectors.

```
{
  "name": "etcd-proxy",
  "profile": "etcd-proxy",
  "metadata": {
    "fleet_metadata": "role=etcd-proxy",
    "etcd_initial_cluster":
"node1=http://node1.example.com:2380,node2=http://node2.example.com:2380,node3=http://node3.example.com:2380"
  }
}
```

For example, a request to `/ignition?mac=52:54:00:89:d8:10` would render the Ignition template in the "etcd" `Profile`, with the machine group's metadata. A request to `/ignition` would match the default group (which has no selectors) and render the Ignition in the "etcd-proxy" Profile. Avoid defining multiple default groups as resolution will not be deterministic.

Reserved selectors

Group selectors can use any key/value pairs you find useful. However, several labels have a defined purpose and will be normalized or parsed specially.

- `uuid` - machine UUID
- `mac` - network interface physical address (normalized MAC address)
- `hostname` - hostname reported by a network boot program
- `serial` - serial reported by a network boot program

Config templates

Profiles can reference various templated configs. Ignition JSON configs can be generated from Container Linux Config template files. Cloud-Config templates files can be used to render a script or Cloud-Config. Generic template files can be used to render arbitrary untyped configs (experimental). Each template may contain Go template elements which will be rendered with machine group metadata, selectors, and query params.

For details and examples:

- Container Linux Config
- Cloud-Config

Variables

Within Container Linux Config templates, Cloud-Config templates, or generic templates, you can use group metadata, selectors, or request-scoped query params. For example, a request `/generic?mac=52-54-00-89-d8-10&foo=some-param&bar=b` would match the `node1.json` machine group shown above. If the group's profile ("etcd") referenced a generic template, the following variables could be used.

```
# Untyped generic config file
# Selector
{{.mac}}               # 52:54:00:89:d8:10 (normalized)
# Metadata
```

```
{{.etcd_name}}          # node1
{{.fleet_metadata}}     # role=etcd,name=node1
# Query
{{.request.query.mac}}  # 52:54:00:89:d8:10 (normalized)
{{.request.query.foo}}  # some-param
{{.request.query.bar}}  # b
# Special Addition
{{.request.raw_query}}  # mac=52:54:00:89:d8:10&foo=some-param&bar=b
```

Note that `.request` is reserved for these purposes so group metadata with data nested under a top level "request" key will be overwritten.

Assets

`matchbox` can serve `-assets-path` static assets at `/assets`. This is helpful for reducing bandwidth usage when serving the kernel and initrd to network booted machines. The default assets-path is `/var/lib/matchbox/assets` or you can pass `-assets-path=""` to disable asset serving.

```
matchbox.foo/assets/
└── coreos
    └── VERSION
        ├── coreos_production_pxe.vmlinuz
        └── coreos_production_pxe_image.cpio.gz
```

For example, a `Profile` might refer to a local asset `/assets/coreos/VERSION/coreos_production_pxe.vmlinuz` instead of `http://stable.release.core-os.net/amd64-usr/VERSION/coreos_production_pxe.vmlinuz`.

See the get-coreos script to quickly download, verify, and place Container Linux assets.

Network

`matchbox` does not implement or exec a DHCP/TFTP server. Read network setup or use the coreos/dnsmasq image if you need a quick DHCP, proxyDHCP, TFTP, or DNS setup.

Going further

- gRPC API Usage
- Metadata
- OpenPGP Signing