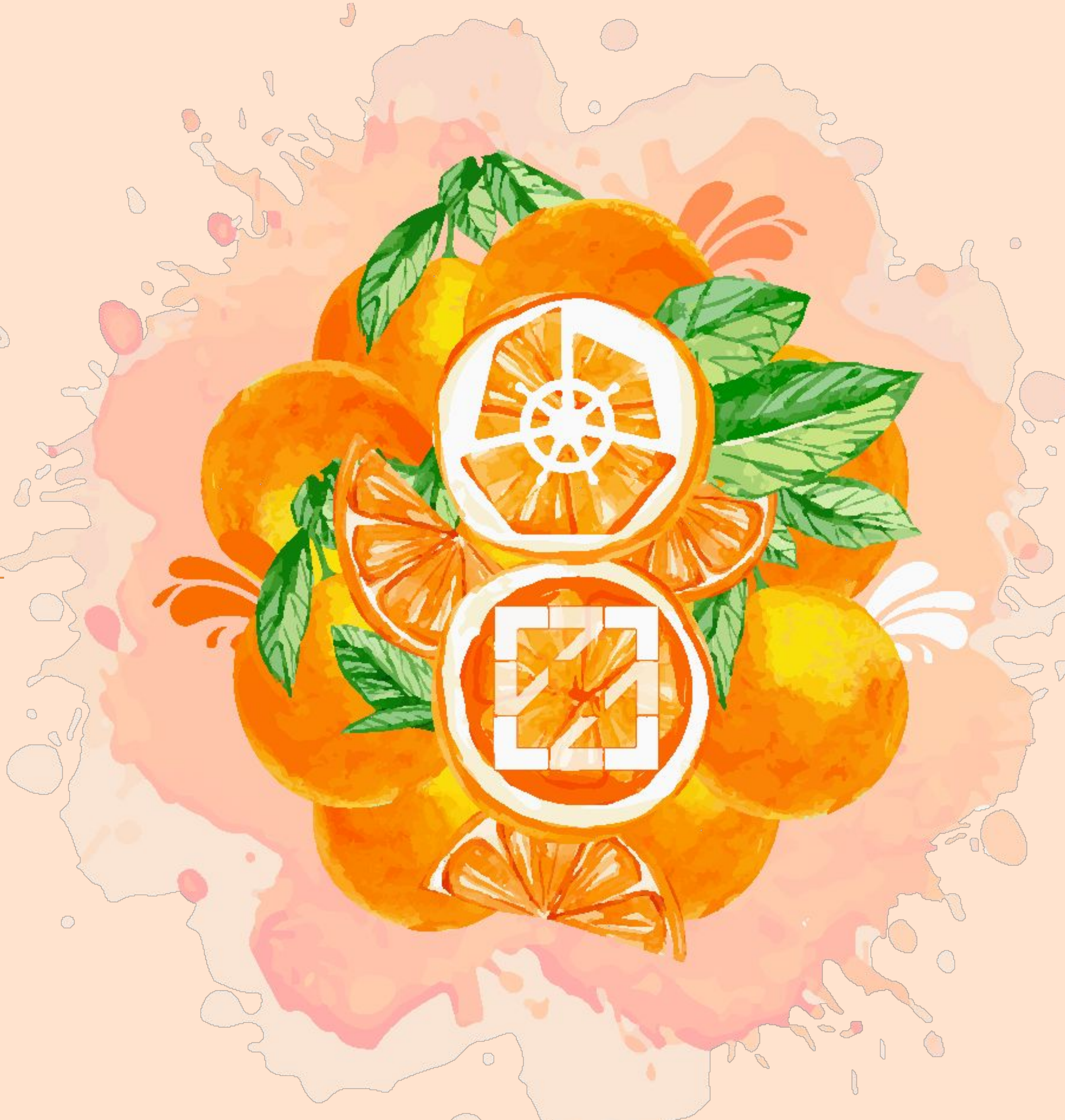
KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA



Build your own Cluster API Provider

The ~~hard~~ easy way

Who are we?

Anusha Hegde

Senior Engineer @ VMWare
Cluster API Provider BYOH Maintainer

Richard Case

Principal Engineer @ Weaveworks
Cluster API Provider AWS Maintainer
Cluster API Provider Microvm Maintainer

Lego & retro gaming addict

Who uses Cluster API (CAPI) already?

Who contributes to a Cluster API Provider?

Who’s thinking of building a Cluster API Provider?

What will we be covering?

- What is CAPI
- Provider Types
- Design
    - First rule of creating a provider……you don’t create a provider. Do you really need one???
    - API design, versioning
    - Controller and CAPI contracts
    - Webhooks
- Patterns
    - Experimental / feature flags
- Dev
    - Tools - kubebuilder, tilt, …
    - Project layout - conventions used by capi & kubebuilder
    - Debugging…tilt/delve is your friend (vscode, goland with tilt)
- Test
    - E2e -CAPI e2e test framework, tradition testing pyramid is out the window
    - Prow when you donate (more later on dnation)
- Community
    - Donation, office hours, slack etc

What will we be covering?

- What is Cluster API
- Different Provider types
- Designing a Provider
- Common Patterns
- Development & Testing
- Community

What is Cluster API? (1/2)

- Built on the premise that “Cluster lifecycle management is difficult”

- Declarative specification of clusters

- Establishes building blocks for higher order functionality
    - Cluster templating
    - Automation of scaling, repair & upgrades
    - Distributing nodes across failure domains
    - Machine health checks to replace unhealthy nodes
    - Managed control planes
    - …..

What is Cluster API? (2/2)

- Designed around interchangeable components via “providers”

- **clusterctl** handles the lifecycle of a CAPI management cluster
    - Cluster templates / flavors are very useful
    - Providers operator being developed…..GitOps friendly provider operations :)

- Community calls every week on Wednesday @ 6pm GMT / 10am PT
    - Separate calls for providers

- For a walkthrough of CAPI see the “lets talk about…” series by Stefan:
    - https://github.com/kubernetes-sigs/cluster-api/discussions/6106

What is Cluster API?

- Built on the premise that “Cluster lifecycle management is difficult”
- Declarative specification of clusters
- **clusterctl** handles the lifecycle of a CAPI management cluster
- Community calls every week on Wednesday @ 6pm GMT / 10am PT
- For a walkthrough of CAPI see the “lets talk about…” series by Stefan:
    - https://github.com/kubernetes-sigs/cluster-api/discussions/6106

What is a Cluster API provider?

A Kubernetes **operator** that implements infrastructure / operating environment specific functionality that is utilized by core Cluster API when managing the lifecycle of a K8s cluster.

The operator implements a contract via its custom resources (i.e. CRDs) depending on the type of provider, which enables interaction between core CAPI and the provider.

Provider Types

- **Infrastructure -** used to provision any infrastructure that is required to create and run a Kubernetes cluster. For example, networking, security groups, virtual or physical host machines

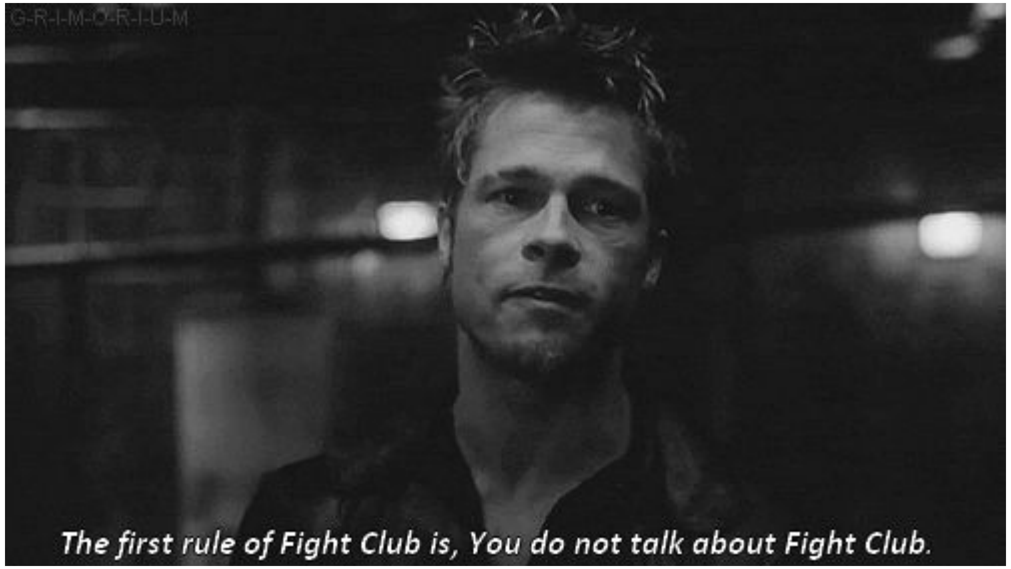
- **Bootstrap -** used to create the “user-data” that is passed to the infrastructure machines that contains the instructions to bootstrap a Kubernetes node on that machine. 2 parts to it:
  - Action: how Kubernetes is bootstrapped (e.g. invoking kubeadm)
  - Format: how the action is encoded and passed to the machine (e.g. cloud-init, ignition)

- **Control plane -** used to control the creation & lifecycle of the Kubernetes control plane. It can utilize resources created by bootstrap and infrastructure providers.
  - Kubeadm control plane is the original
  - Managed Kubernetes (i.e. EKS, AKS) implementations - no nodes

First rule of creating a provider…

…you don’t need to create a provider!

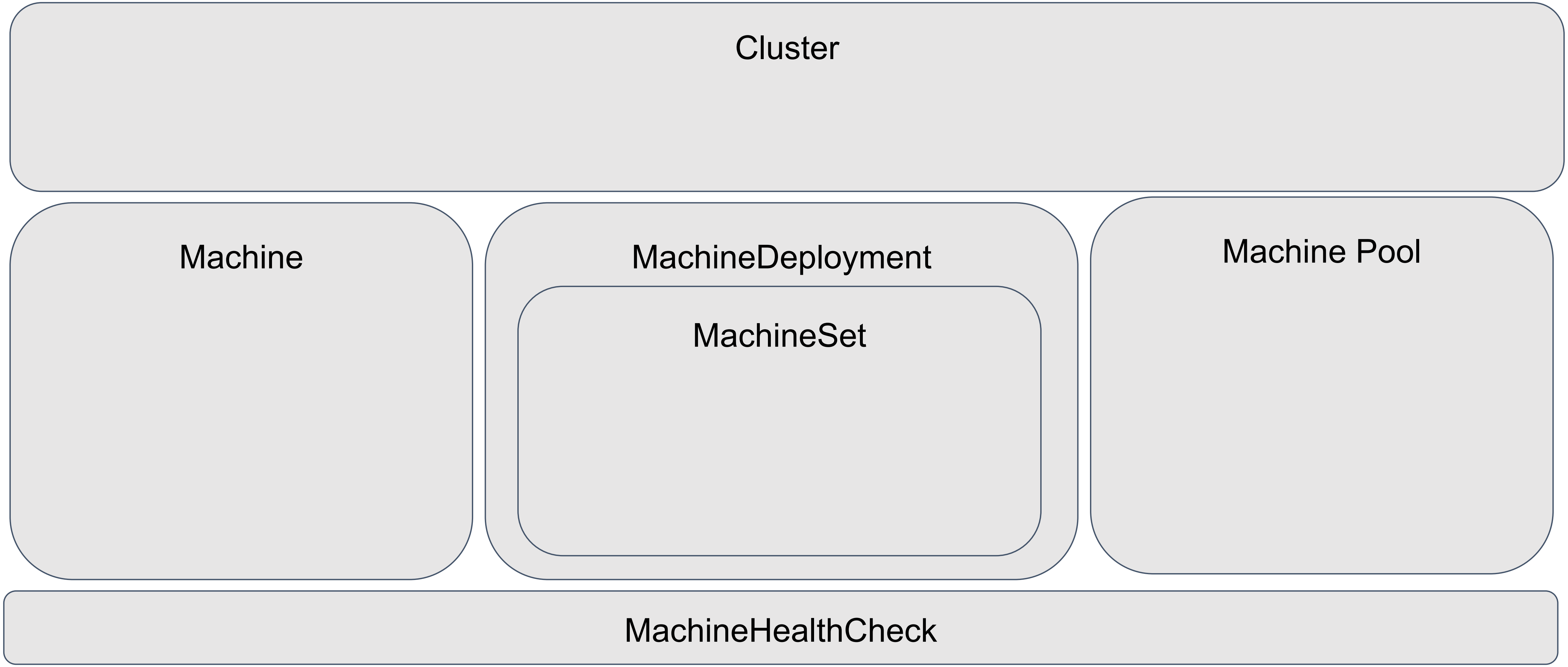
(hopefully)


G-R-I-M-O-R-I-U-M
The first rule of Fight Club is, You do not talk about Fight Club.

What constitutes a Cluster API provider?

- A Kubernetes operator (a.k.a controller manager)
  - CRDs
  - Controllers that reconcile the CRDs
- k8s resources to deploy the controller
  - Plain old yaml
  - (Optional) tokens that will be replaced an installation time
  - Kustomize
- Metadata / repo layout

Core



Cluster

Machine

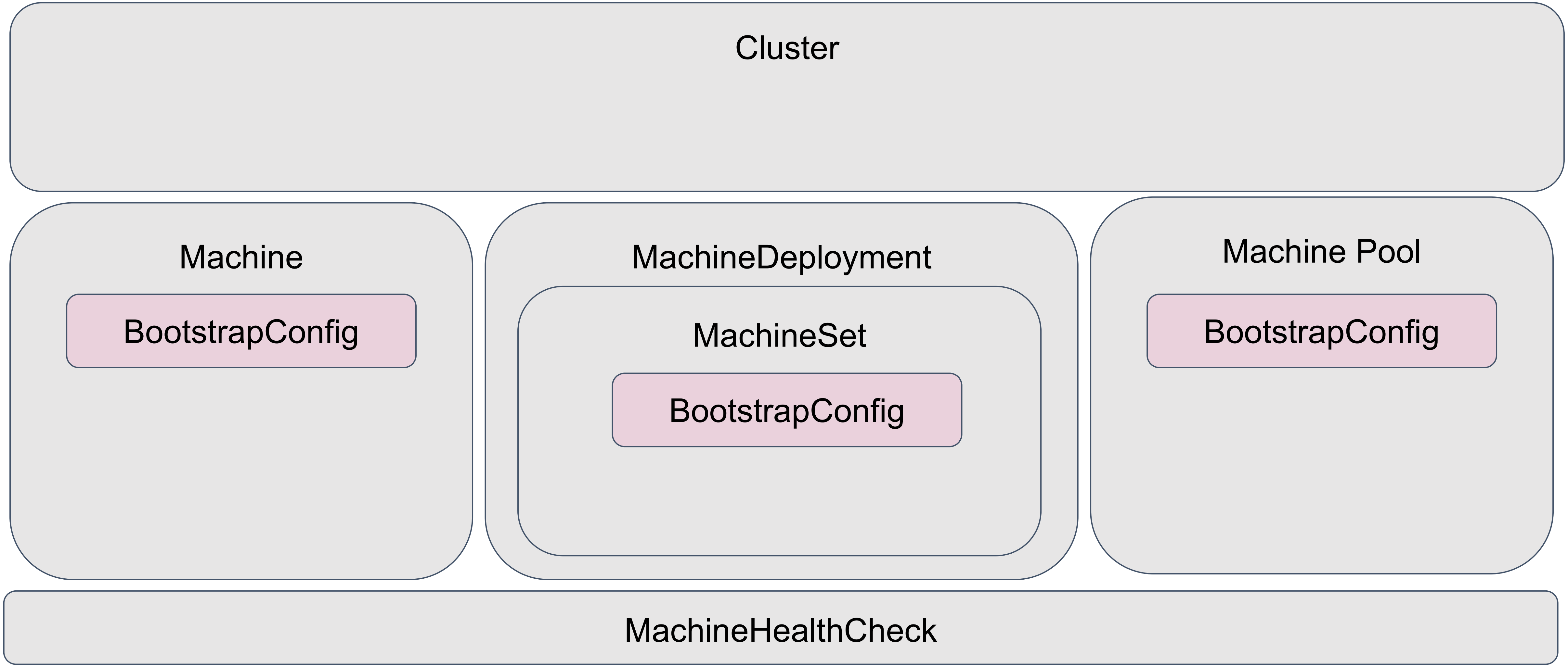
MachineDeployment

MachineSet

Machine Pool

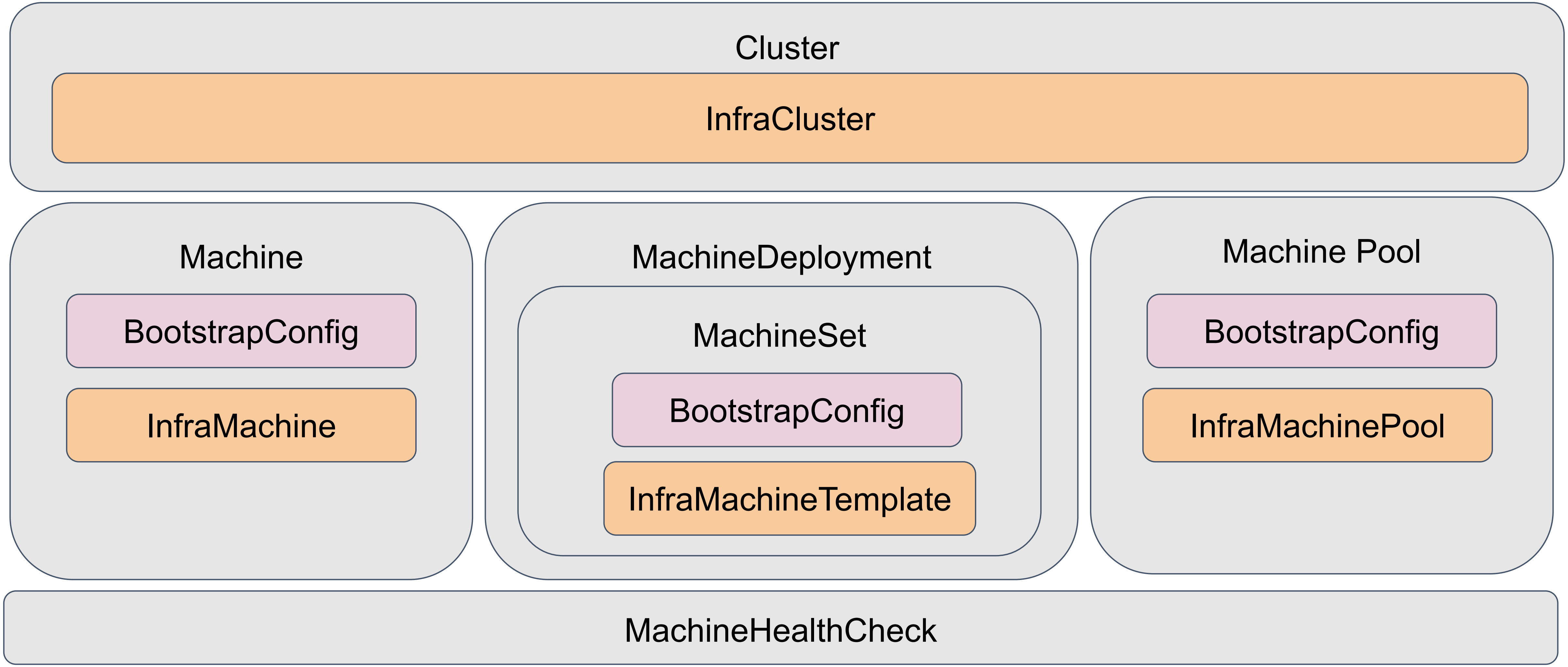
MachineHealthCheck

Bootstrap

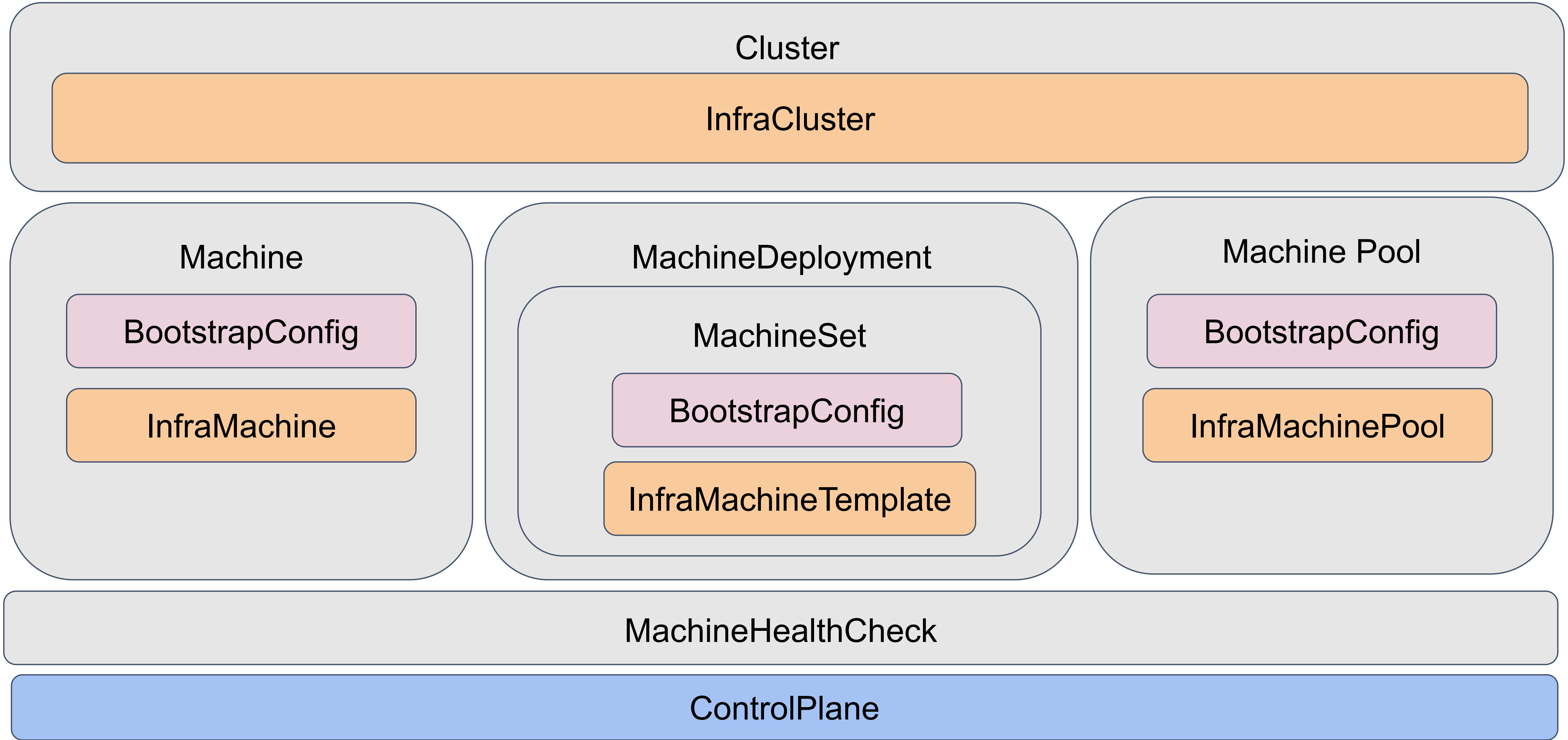


Cluster
Machine
BootstrapConfig
MachineDeployment
MachineSet
BootstrapConfig
Machine Pool
BootstrapConfig
MachineHealthCheck

Infrastructure



Control Plane



Cluster

InfraCluster

Machine

BootstrapConfig

InfraMachine

MachineDeployment

MachineSet

BootstrapConfig

InfraMachineTemplate

Machine Pool

BootstrapConfig

InfraMachinePool

MachineHealthCheck

ControlPlane

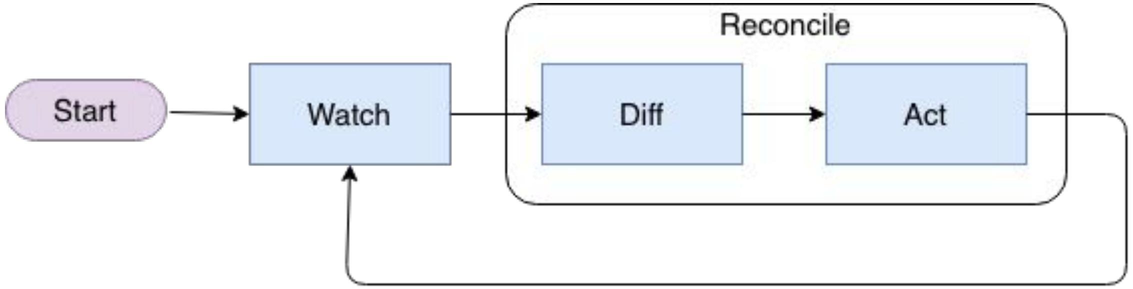
What is an operator?

- A way to create, manage and configure complex applications in Kubernetes.
- Codifies the steps a human would do to deploy and operate a complex application
  - For example, the steps to create Kubernetes cluster involved creating infrastructure, bootstrapping k8s, managing version upgrades.
- Surface to user via declarative API (i.e. CRDs)
- Contains one or more controllers that understand & reconcile the CRDs

What is a controller?

“ A controller is a control loop that watches the desired state of the cluster through the API server and makes changes attempting to move the current state towards the desired state. “

Control Loop



- Watch - for changes in custom resource(s)
- Diff - work out the difference between desired & actual state
- Act - take action to remediate the difference (if any)

…..And repeat!

What provider type do you need?

- Do you operate a cloud / baremetal service?
  - You’ll need an Infrastructure provider

- Do you want a different way to bootstrap Kubernetes instead of Kubeadm?
  - You’ll need a bootstrap provider and maybe a control plane provider

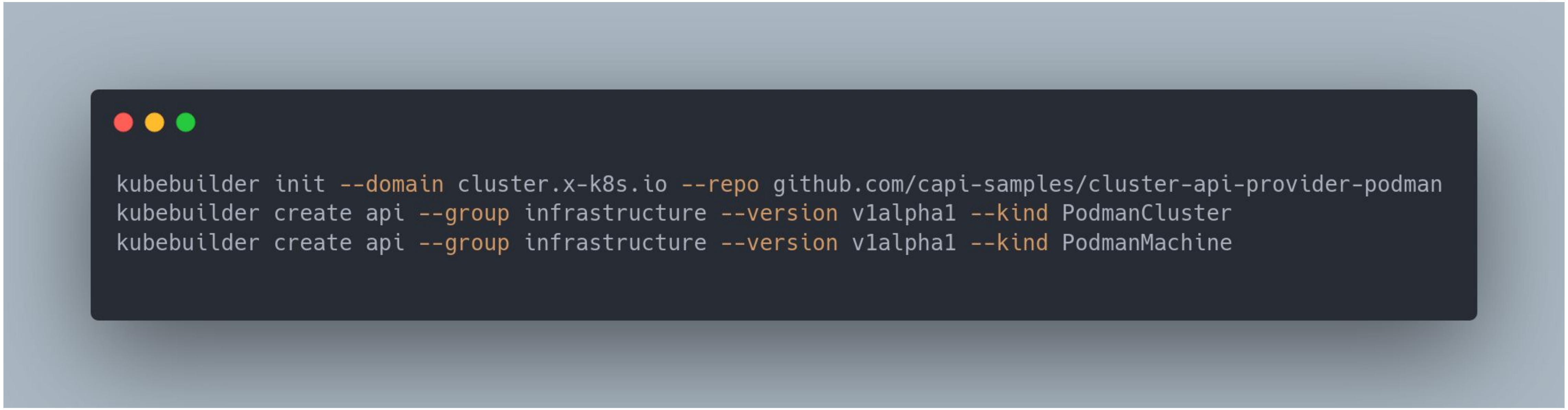
- Do you have a hosted Kubernetes control plane service?
  - You may need a control plane provider

- Do you want to use virtualization technology like KVM or vSphere?
  - You may be covered by the vSphere, Microvm or Kubevirt providers
  - If not then may need to create a infrastructure provider

- Do you want to provision your own infrastructure and get CAPI to manage Kubernetes?
  - You may be covered by one of the existing providers that allow you to prong your own infrastructure (like CAPA, CAPZ)
  - You may be covered by the “bring your own host” provider
  - If not then you may need any of the 3 types of provider

Scaffolding the provider (1/2)

- Kubebuilder and controller-runtime are your friends:



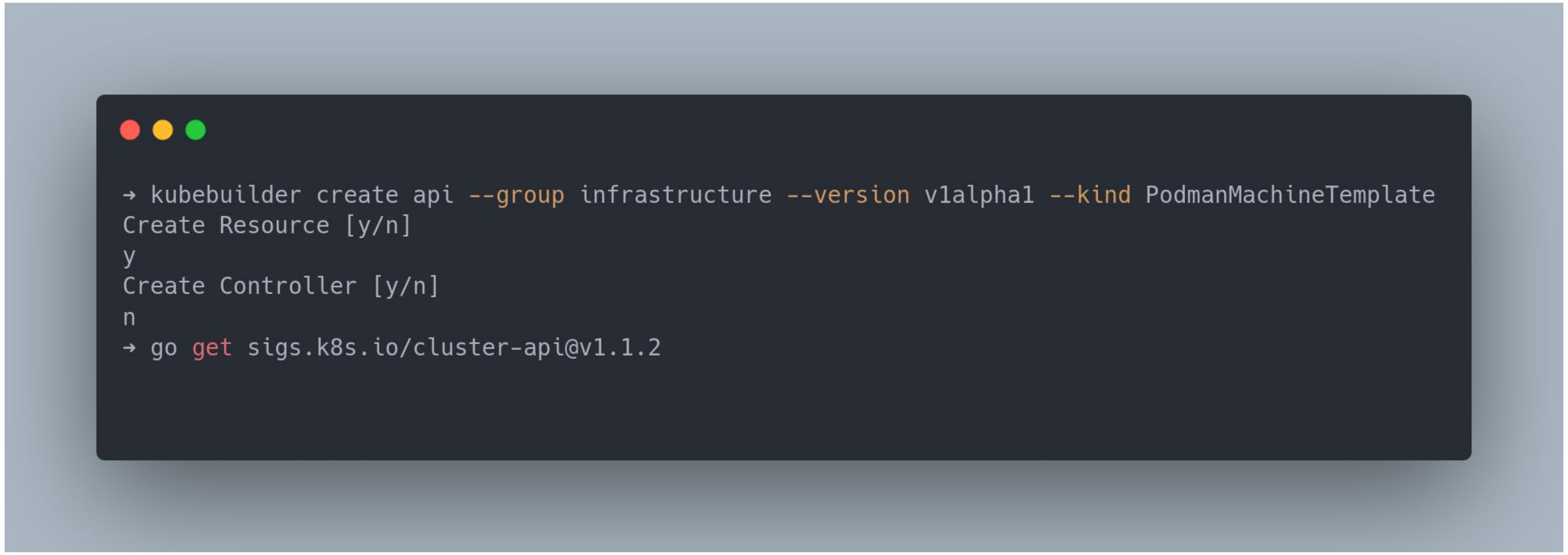
```
kubebuilder init --domain cluster.x-k8s.io --repo github.com/capi-samples/cluster-api-provider-podman
kubebuilder create api --group infrastructure --version v1alpha1 --kind PodmanCluster
kubebuilder create api --group infrastructure --version v1alpha1 --kind PodmanMachine
```

Important parts:

- **–domain cluster.x-k8s.io** - this generally used as the domain part of the GVK by providers
- **–group infrastructure** - this by convention indicates that the CRD relates to an infrastructure provider
- **–version v1alpha1** - providers need to follow the Kubernetes API versioning standard and the guarantees these provide to the user
- **–kind PodmanCluster/Machine** - by convention, a provider uses a consistent prefix on the CRD names

Scaffolding the provider (2/2)

- We also need a machine template API type **<u>BUT no controller</u>**
- Add a reference to CAPI so we can use their API definitions / utility functions



```
➜ kubebuilder create api --group infrastructure --version v1alpha1 --kind PodmanMachineTemplate
Create Resource [y/n]
y
Create Controller [y/n]
n
➜ go get sigs.k8s.io/cluster-api@v1.1.2
```

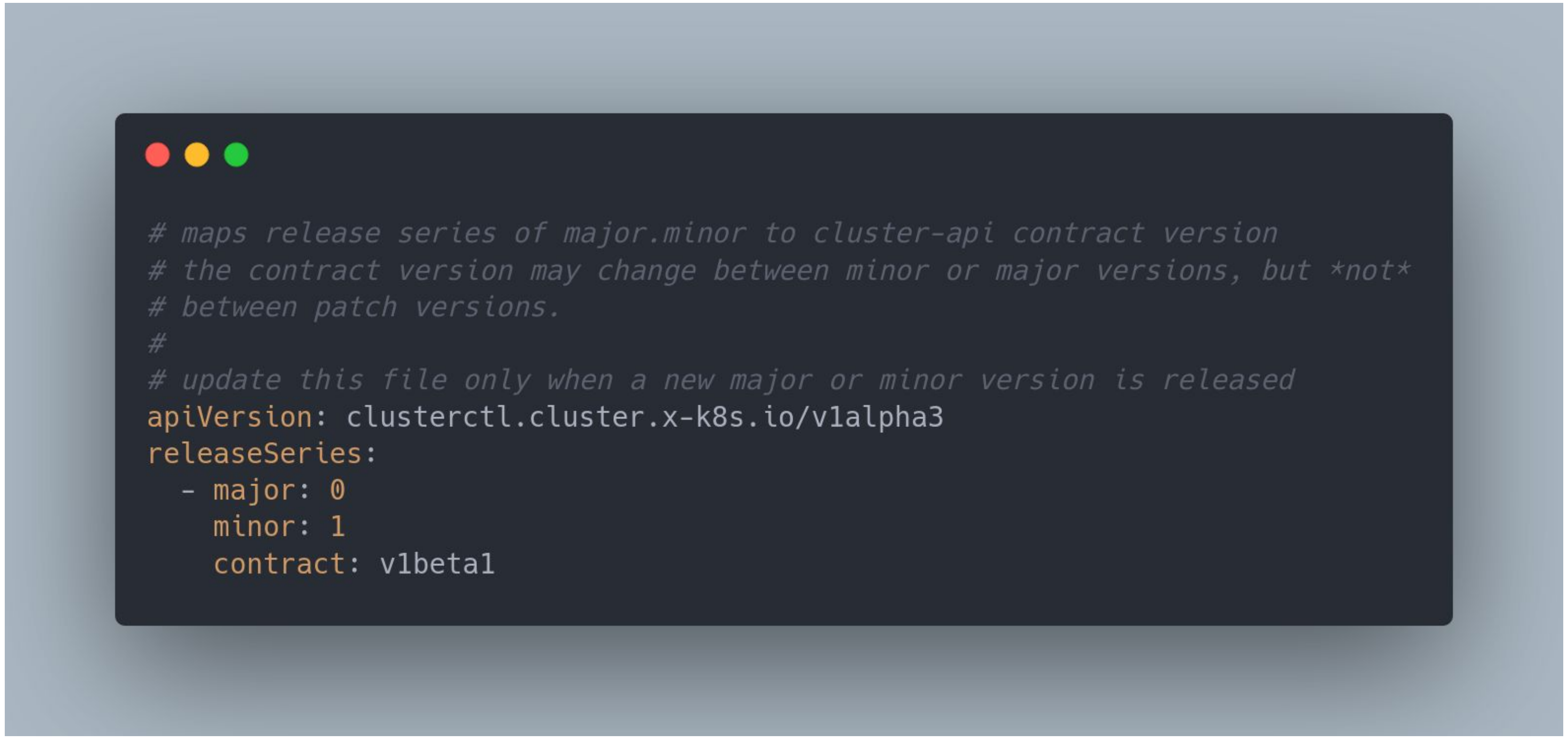
Why do we not need a controller for the machine template?

- It's used as a template to create new instances of **PodmanMachine**
- **PodmanMachine** has a controller to handle reconciliation

Provider Metadata

A provider must specify which versions are compatible with which CAPI API version.

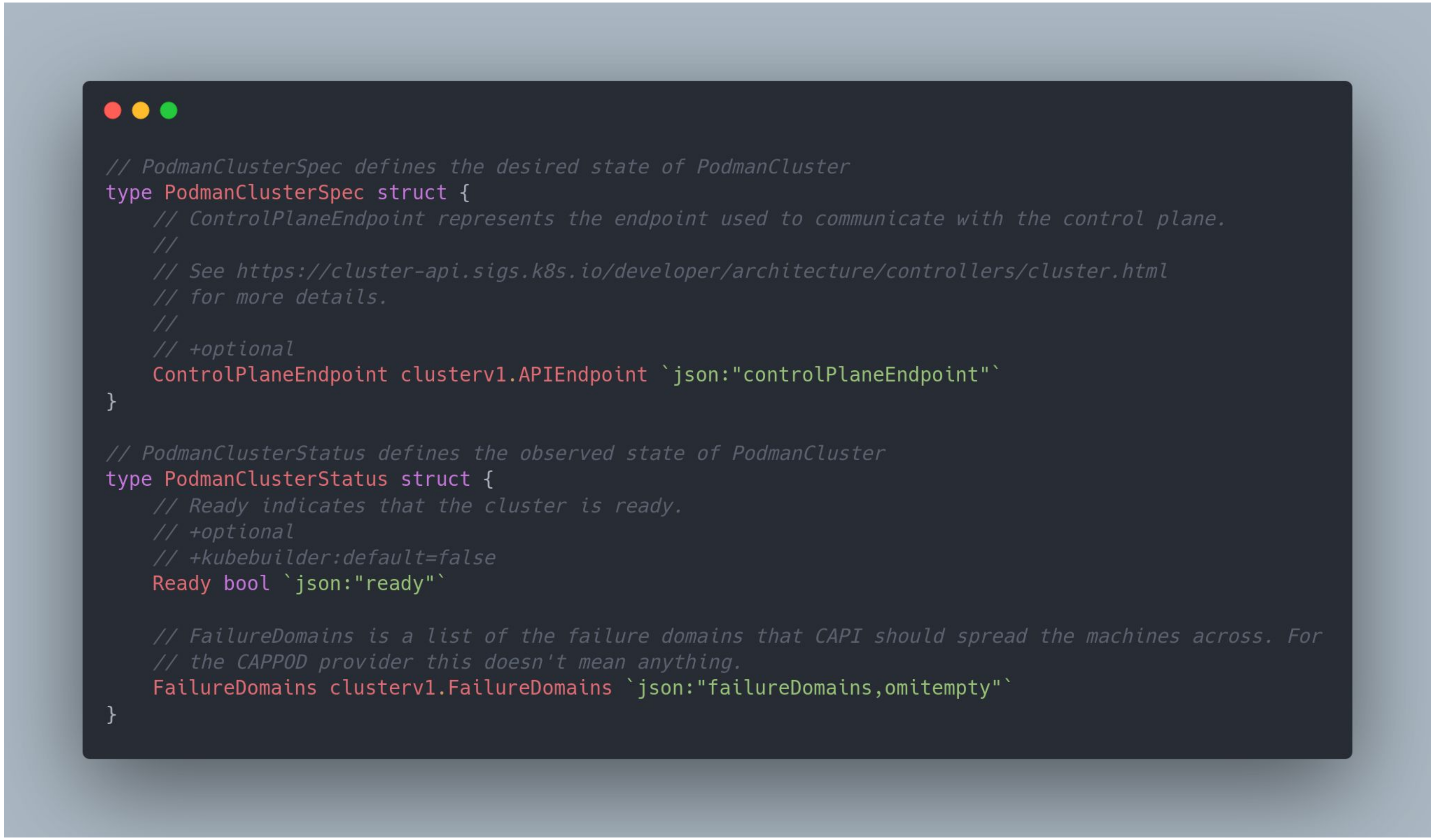
- One of the requirements to be installable via **clusterctl init**
- Create a metadata.yaml file in the root of the repo



```
# maps release series of major.minor to cluster-api contract version
# the contract version may change between minor or major versions, but *not*
# between patch versions.
#
# update this file only when a new major or minor version is released
apiVersion: clusterctl.cluster.x-k8s.io/v1alpha3
releaseSeries:
  - major: 0
    minor: 1
    contract: v1beta1
```

Define API for your provider (1/2)

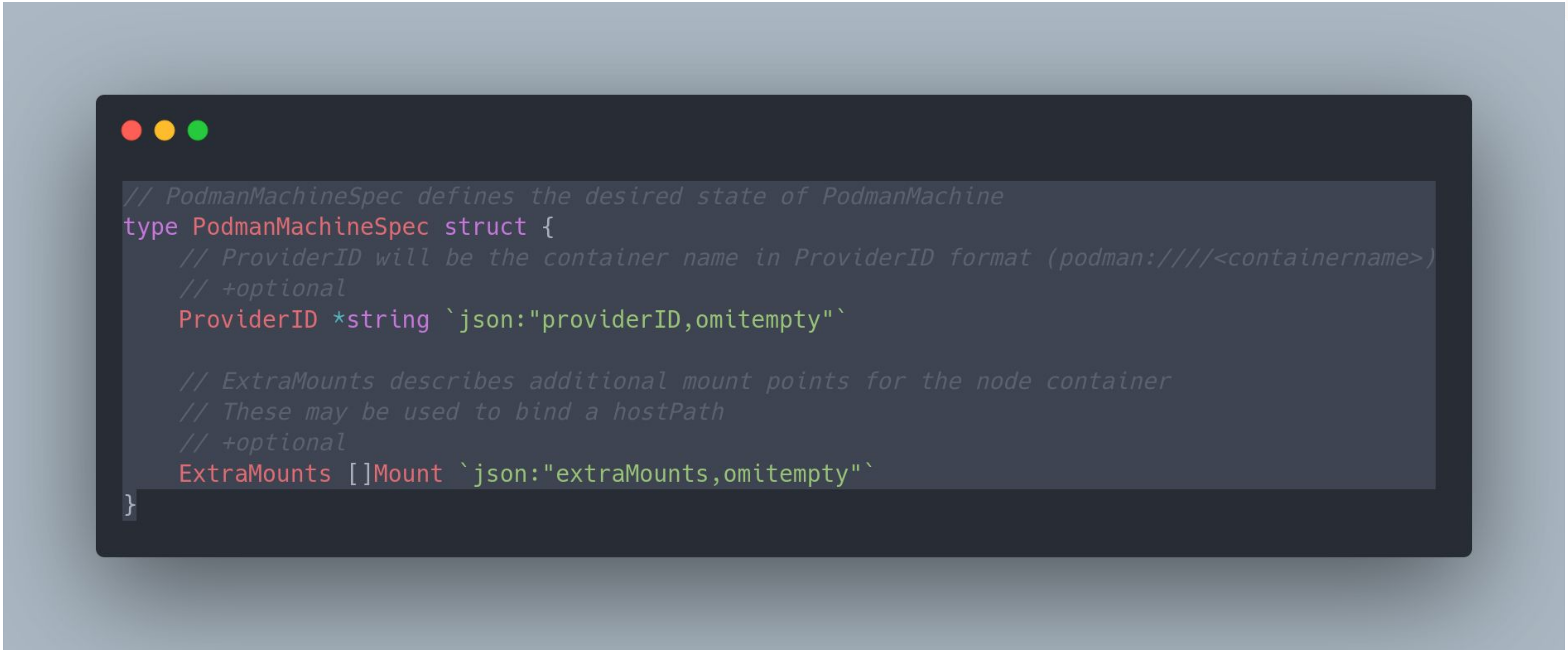
- Add fields to the **Spec** & **Status** of your API types to conform to your provider types contract
  - CAPI documentation helps: https://cluster-api.sigs.k8s.io/developer/providers/implementers.html



```go
// PodmanClusterSpec defines the desired state of PodmanCluster
type PodmanClusterSpec struct {
    // ControlPlaneEndpoint represents the endpoint used to communicate with the control plane.
    //
    // See https://cluster-api.sigs.k8s.io/developer/architecture/controllers/cluster.html
    // for more details.
    //
    // +optional
    ControlPlaneEndpoint clusterv1.APIEndpoint `json:"controlPlaneEndpoint"`
}

// PodmanClusterStatus defines the observed state of PodmanCluster
type PodmanClusterStatus struct {
    // Ready indicates that the cluster is ready.
    // +optional
    // +kubebuilder:default=false
    Ready bool `json:"ready"`

    // FailureDomains is a list of the failure domains that CAPI should spread the machines across. For
    // the CAPPOD provider this doesn't mean anything.
    FailureDomains clusterv1.FailureDomains `json:"failureDomains,omitempty"`
}
```

Define API for your provider (2/2)

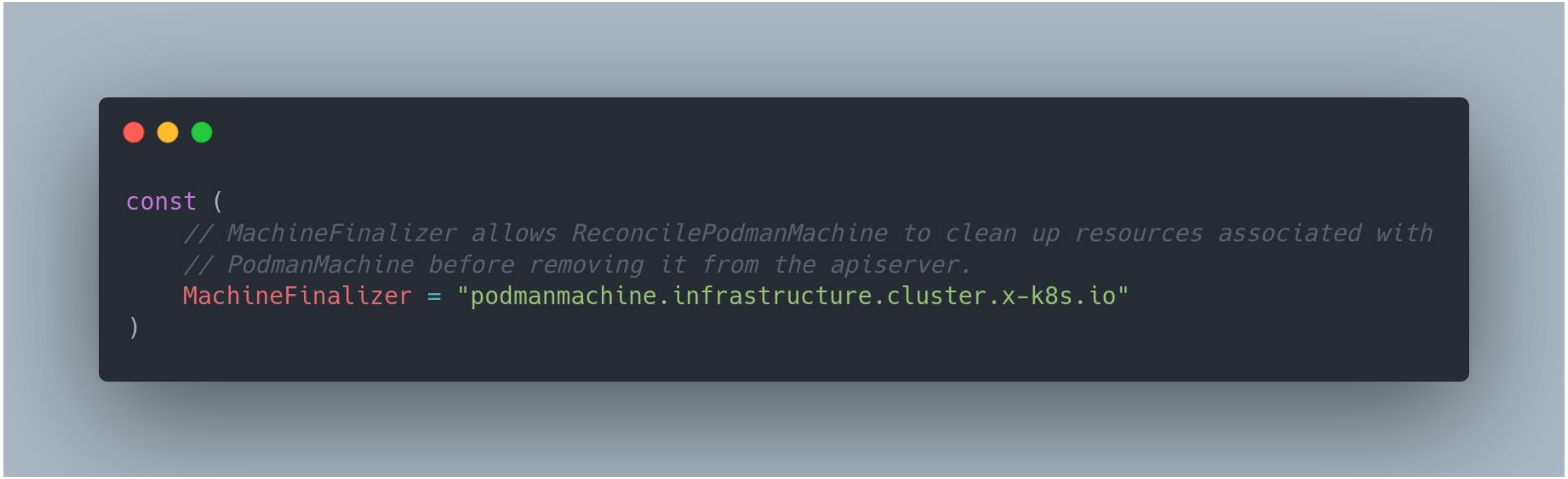
- Add custom fields to **the Spec** & **Status** of your API types that are specific to your provider



```go
// PodmanMachineSpec defines the desired state of PodmanMachine
type PodmanMachineSpec struct {
    // ProviderID will be the container name in ProviderID format (podman:////<containername>)
    // +optional
    ProviderID *string `json:"providerID,omitempty"`

    // ExtraMounts describes additional mount points for the node container
    // These may be used to bind a hostPath
    // +optional
    ExtraMounts []Mount `json:"extraMounts,omitempty"`
}
```

Add finalizers

- **If your provider creates external resources you will need to define finalizers**
  - A finalizer allows the controller to clean up external resources before allowing the API type to be deleted from API server.
  - See docs: https://book.kubebuilder.io/reference/using-finalizers.html

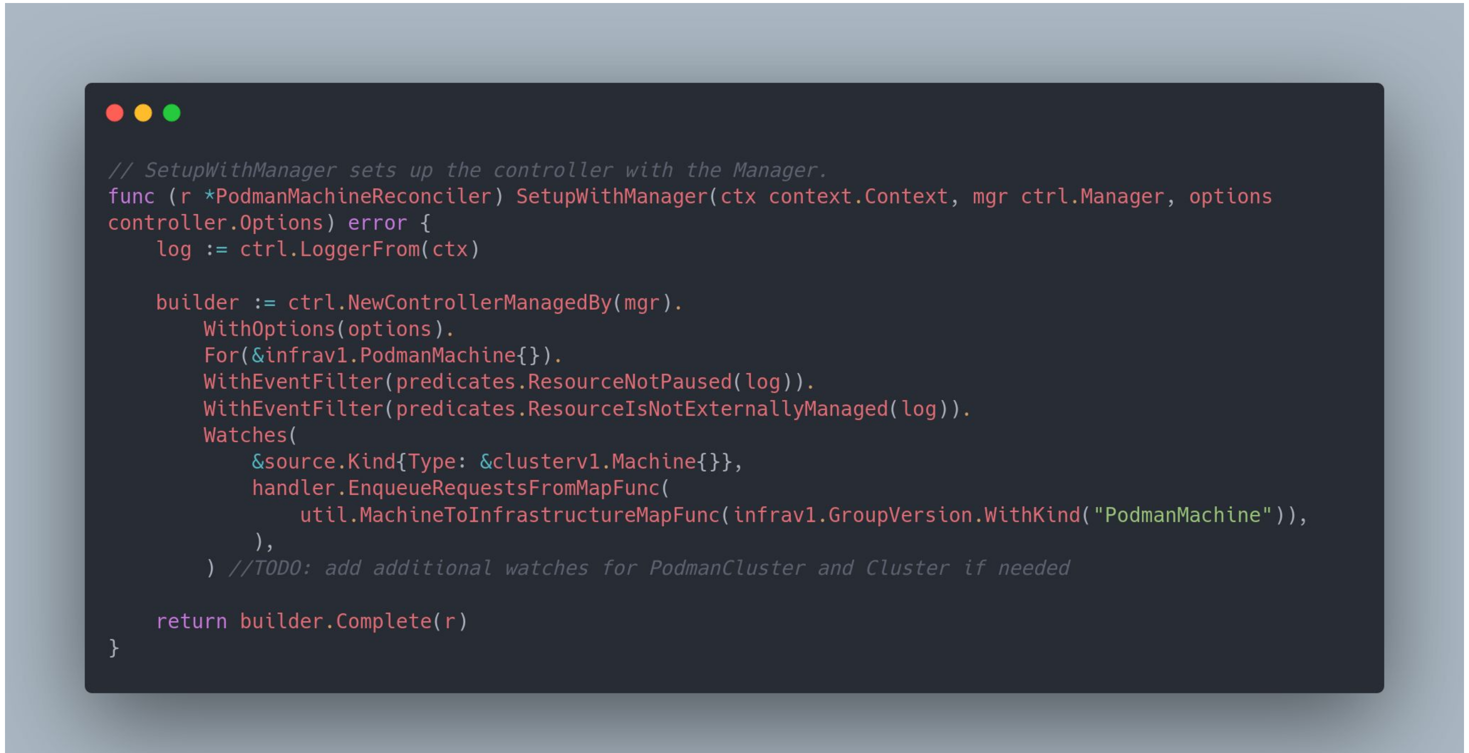


```
const (
    // MachineFinalizer allows ReconcilePodmanMachine to clean up resources associated with
    // PodmanMachine before removing it from the apiserver.
    MachineFinalizer = "podmanmachine.infrastructure.cluster.x-k8s.io"
)
```

- The controllers will add and remove the finalizers

Implement controllers for your API types (1/2)

- Kubebuilder will have created an “empty” controller with
  - **Reconcile** function
  - Controller setup to watch its CRD type
- Setup tasks that you need to do:
  - Watch companion CRDs from CAPI
  - Ensure reconciliation doesn’t occur if paused or if the resource is externally managed



```go
// SetupWithManager sets up the controller with the Manager.
func (r *PodmanMachineReconciler) SetupWithManager(ctx context.Context, mgr ctrl.Manager, options
controller.Options) error {
    log := ctrl.LoggerFrom(ctx)

    builder := ctrl.NewControllerManagedBy(mgr).
        WithOptions(options).
        For(&infrav1.PodmanMachine{}).
        WithEventFilter(predicates.ResourceNotPaused(log)).
        WithEventFilter(predicates.ResourceIsNotExternallyManaged(log)).
        Watches(
            &source.Kind{Type: &clusterv1.Machine{}},
            handler.EnqueueRequestsFromMapFunc(
                util.MachineToInfrastructureMapFunc(infrav1.GroupVersion.WithKind("PodmanMachine")),
            ),
        ) //TODO: add additional watches for PodmanCluster and Cluster if needed

    return builder.Complete(r)
}
```

Implement controllers for your API types (2/2)

For **Reconcile** the following pattern is generally used:

1. Get the instance of the API type being reconciled
2. Get the owning CAPI type (i.e. if we reconciling **PodmanMachine** then get **Machine**)
3. If we don’t have the machine then exit (owner reference isn’t set yet)
4. Optionally, get the owning **Cluster** and Infra Cluster (i.e. **PodManCluster**)
5. If instance has a deletion timestamp, then in **reconcileDelete**:
    a. do any actions to delete
    b. remove finalizer and save
6. If instance has no deletion timestamp, then in **reconcileNormal**:
    a. Add finalizer to instance and save
    b. do any actions to create OR update

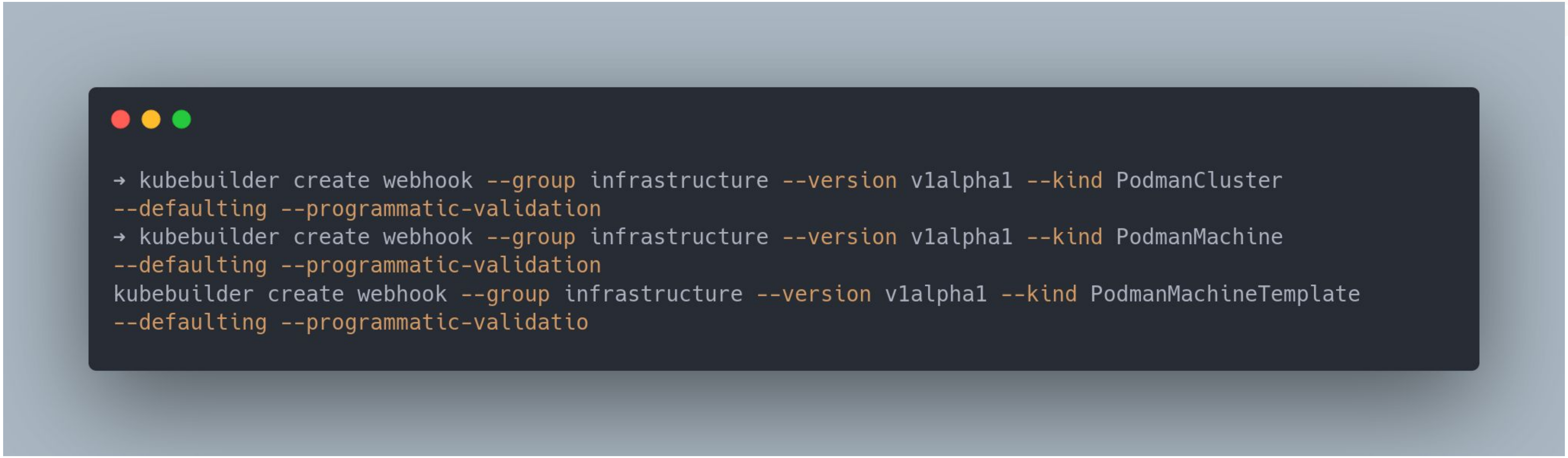
Owner Reference

When building a provider you should set ownerReference

- A link to a resource that is the owner
  - Example: Deployment owns Pods
  - Example: Cluster owns PodmanCluster
- Used heavily in Cluster API
- Implemented via the **metadata.ownerReference** field
- If the owner is deleted then either:
  - Cascading deletion (controlled via policy)
  - Orphaned resources

Webhooks

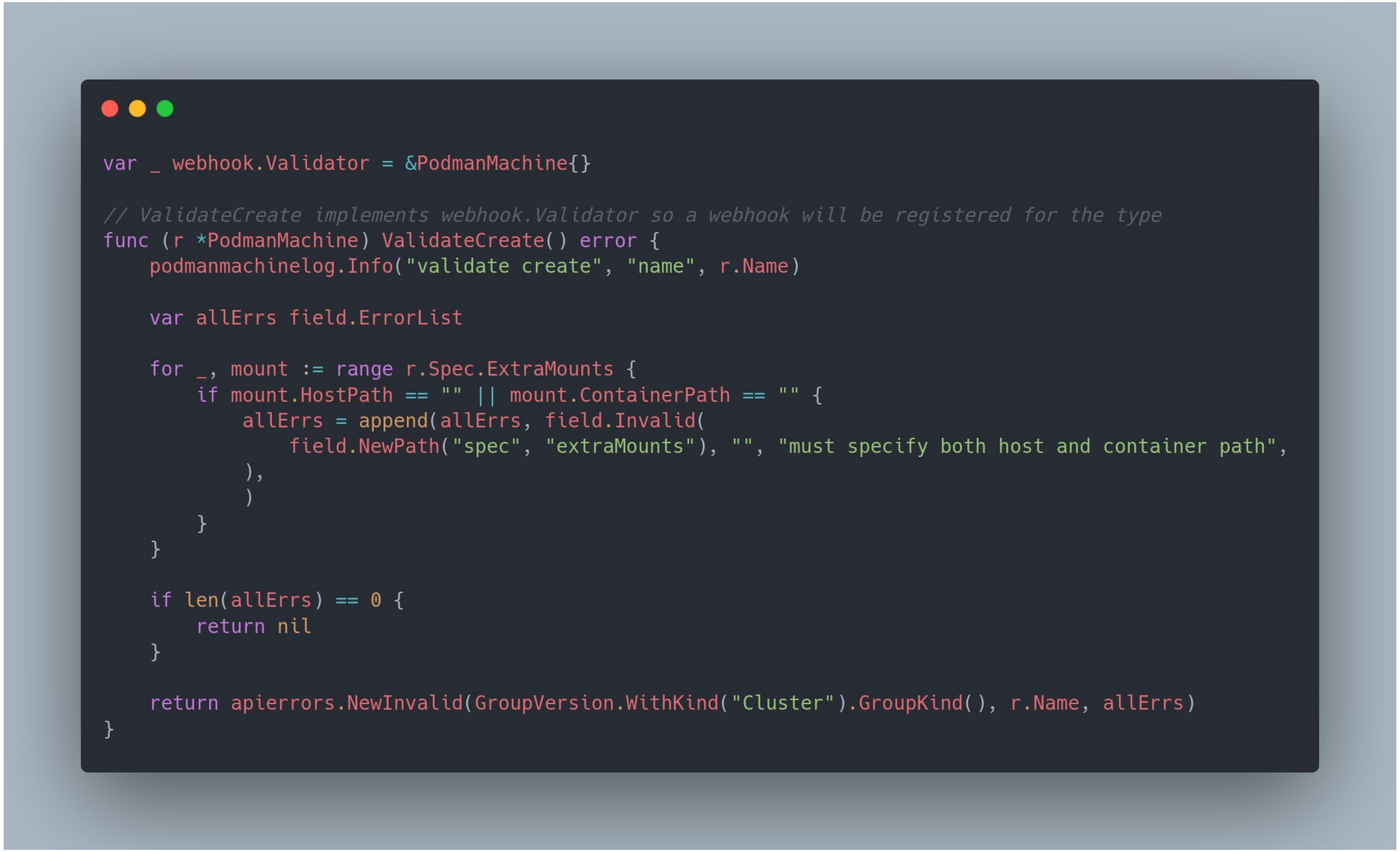
If you need custom logic for defaults or validation you can create webhooks:



```
➜ kubebuilder create webhook --group infrastructure --version v1alpha1 --kind PodmanCluster
--defaulting --programmatic-validation
➜ kubebuilder create webhook --group infrastructure --version v1alpha1 --kind PodmanMachine
--defaulting --programmatic-validation
kubebuilder create webhook --group infrastructure --version v1alpha1 --kind PodmanMachineTemplate
--defaulting --programmatic-validatio
```

- Webhooks are Kubernetes Admission controllers
- This will skaffold both a **defaulting** & **validating** webhook
  - It's your responsibility to fill in the logic
- Defaulting webhook should only be used where kubebuilder defaults are not sufficient

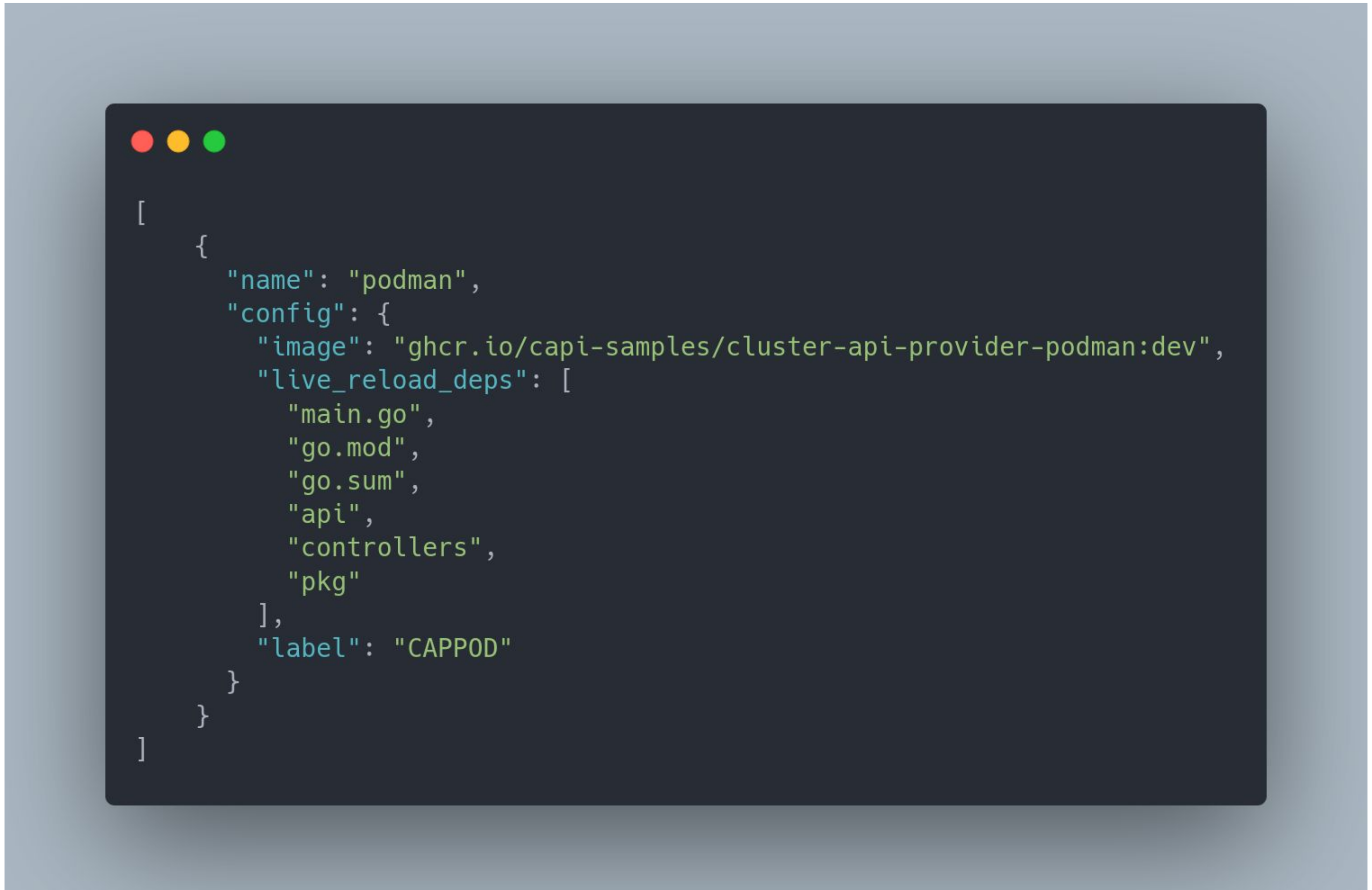
Webhook - Implementation



```go
var _ webhook.Validator = &PodmanMachine{}

// ValidateCreate implements webhook.Validator so a webhook will be registered for the type
func (r *PodmanMachine) ValidateCreate() error {
    podmanmachinelog.Info("validate create", "name", r.Name)

    var allErrs field.ErrorList

    for _, mount := range r.Spec.ExtraMounts {
        if mount.HostPath == "" || mount.ContainerPath == "" {
            allErrs = append(allErrs, field.Invalid(
                field.NewPath("spec", "extraMounts"), "", "must specify both host and container path",
            ),
            )
        }
    }

    if len(allErrs) == 0 {
        return nil
    }

    return apierrors.NewInvalid(GroupVersion.WithKind("Cluster").GroupKind(), r.Name, allErrs)
}
```

Local testing / development (1/2)

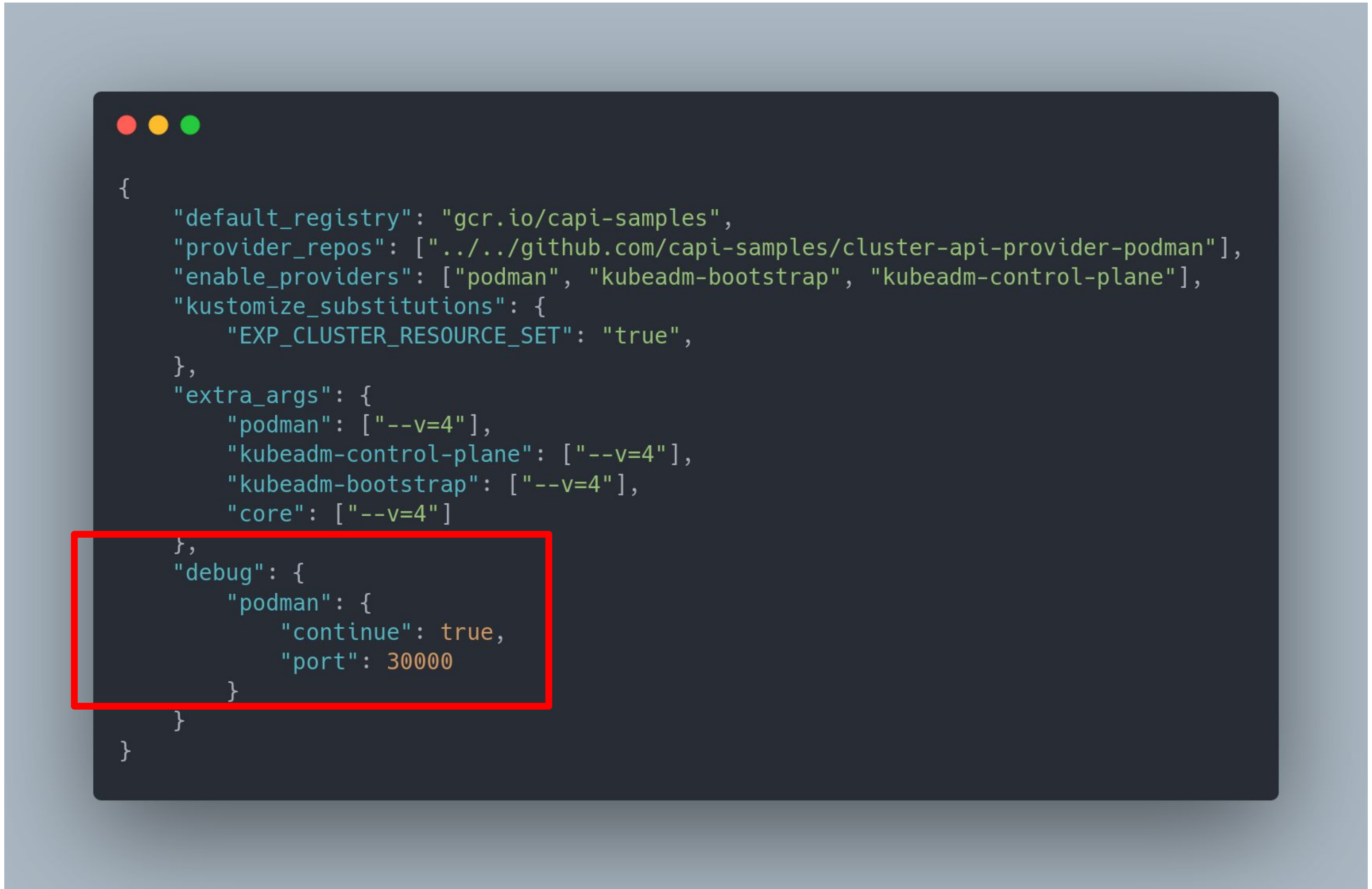
- Developing and debugging operators in Kubernetes can be painful.
- Tilt will save you a lot of time, pain and tears!
  - We need to tell Tilt about our provider via the **tilt-provider.json** file in repo root



```
[
    {
      "name": "podman",
      "config": {
        "image": "ghcr.io/capi-samples/cluster-api-provider-podman:dev",
        "live_reload_deps": [
          "main.go",
          "go.mod",
          "go.sum",
          "api",
          "controllers",
          "pkg"
        ],
        "label": "CAPPOD"
      }
    }
]
```

Local testing / development (2/2)

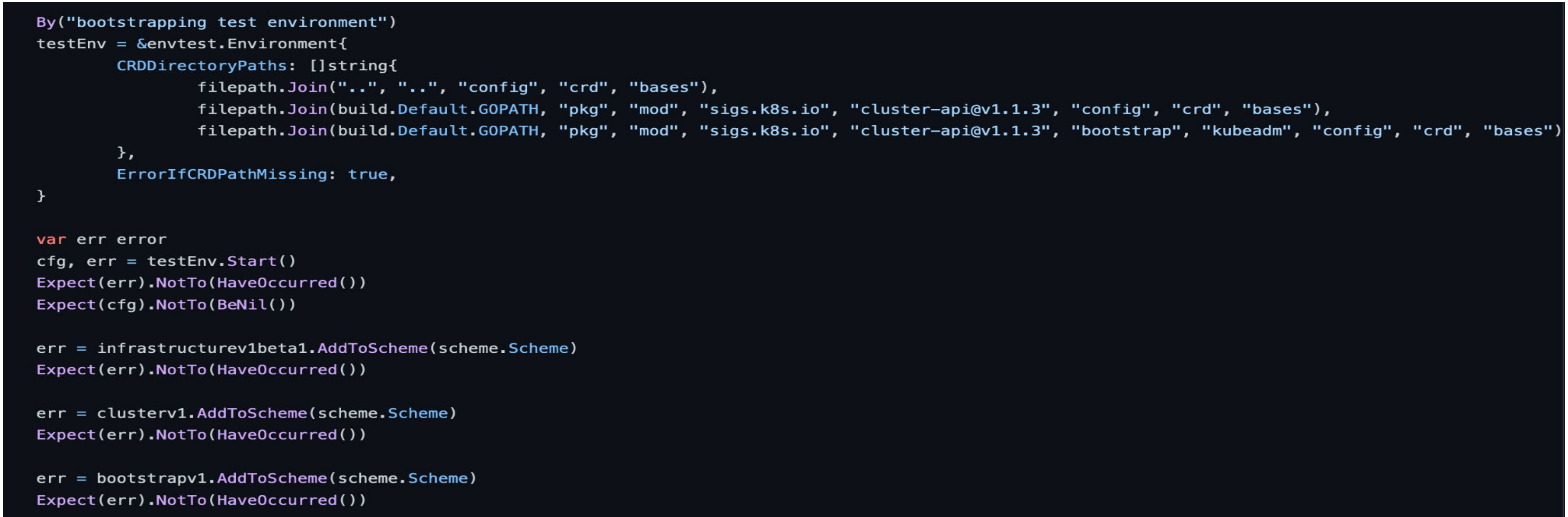
- We can then follow the instructions from the CAPI docs to configure Tilt:
  - https://cluster-api.sigs.k8s.io/developer/tilt.html



```
{
    "default_registry": "gcr.io/capi-samples",
    "provider_repos": ["../../github.com/capi-samples/cluster-api-provider-podman"],
    "enable_providers": ["podman", "kubeadm-bootstrap", "kubeadm-control-plane"],
    "kustomize_substitutions": {
        "EXP_CLUSTER_RESOURCE_SET": "true",
    },
    "extra_args": {
        "podman": ["--v=4"],
        "kubeadm-control-plane": ["--v=4"],
        "kubeadm-bootstrap": ["--v=4"],
        "core": ["--v=4"]
    },
    "debug": {
        "podman": {
            "continue": true,
            "port": 30000
        }
    }
}
```

Testing

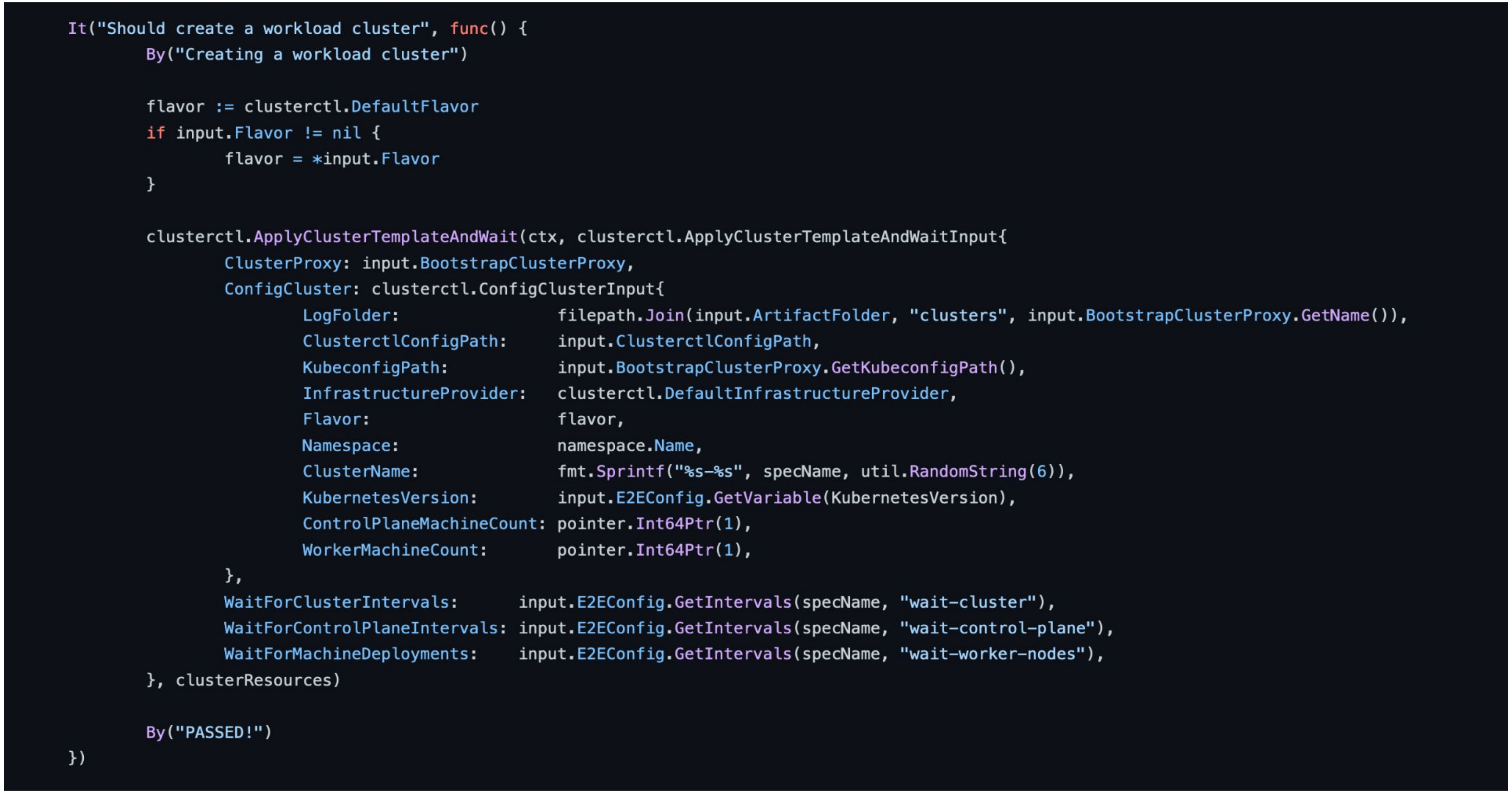
- Unit and integration tests are up to the provider implementers to follow the frameworks / strategy of their choice
- Envtest can be used for unit and integration tests
    - https://github.com/kubernetes-sigs/controller-runtime/tree/master/pkg/envtest
- With envtest, it is possible to interact with your provider like a real cluster
    - You can create / update CRDs and controllers can take action on events



```
By("bootstrapping test environment")
testEnv = &envtest.Environment{
        CRDDirectoryPaths: []string{
                filepath.Join("..", "..", "config", "crd", "bases"),
                filepath.Join(build.Default.GOPATH, "pkg", "mod", "sigs.k8s.io", "cluster-api@v1.1.3", "config", "crd", "bases"),
                filepath.Join(build.Default.GOPATH, "pkg", "mod", "sigs.k8s.io", "cluster-api@v1.1.3", "bootstrap", "kubeadm", "config", "crd", "bases")
        },
        ErrorIfCRDPathMissing: true,
}

var err error
cfg, err = testEnv.Start()
Expect(err).NotTo(HaveOccurred())
Expect(cfg).NotTo(BeNil())

err = infrastructurev1beta1.AddToScheme(scheme.Scheme)
Expect(err).NotTo(HaveOccurred())

err = clusterv1.AddToScheme(scheme.Scheme)
Expect(err).NotTo(HaveOccurred())

err = bootstrapv1.AddToScheme(scheme.Scheme)
Expect(err).NotTo(HaveOccurred())
```

Testing Continued

- CAPI provides e2e framework - most of the code is reusable



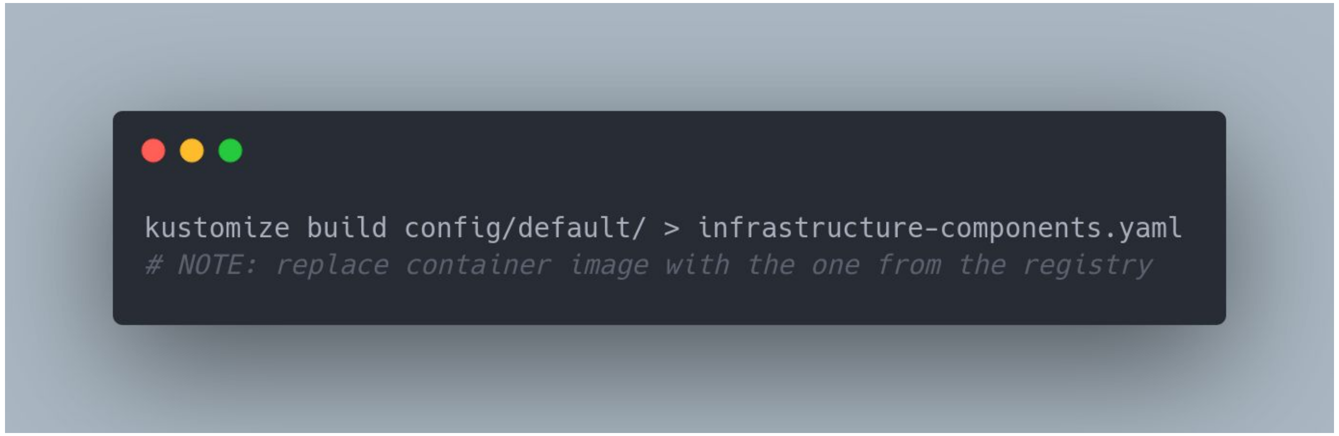
```
It("Should create a workload cluster", func() {
        By("Creating a workload cluster")

        flavor := clusterctl.DefaultFlavor
        if input.Flavor != nil {
                flavor = *input.Flavor
        }

        clusterctl.ApplyClusterTemplateAndWait(ctx, clusterctl.ApplyClusterTemplateAndWaitInput{
                ClusterProxy: input.BootstrapClusterProxy,
                ConfigCluster: clusterctl.ConfigClusterInput{
                        LogFolder:                filepath.Join(input.ArtifactFolder, "clusters", input.BootstrapClusterProxy.GetName()),
                        ClusterctlConfigPath:     input.ClusterctlConfigPath,
                        KubeconfigPath:           input.BootstrapClusterProxy.GetKubeconfigPath(),
                        InfrastructureProvider:   clusterctl.DefaultInfrastructureProvider,
                        Flavor:                   flavor,
                        Namespace:                namespace.Name,
                        ClusterName:              fmt.Sprintf("%s-%s", specName, util.RandomString(6)),
                        KubernetesVersion:        input.E2EConfig.GetVariable(KubernetesVersion),
                        ControlPlaneMachineCount: pointer.Int64Ptr(1),
                        WorkerMachineCount:       pointer.Int64Ptr(1),
                },
                WaitForClusterIntervals:      input.E2EConfig.GetIntervals(specName, "wait-cluster"),
                WaitForControlPlaneIntervals: input.E2EConfig.GetIntervals(specName, "wait-control-plane"),
                WaitForMachineDeployments:    input.E2EConfig.GetIntervals(specName, "wait-worker-nodes"),
        }, clusterResources)

        By("PASSED!")
})
```

Releasing

To be installable via **clusterctl init** you must:

- Publish your provider as a container to a registry
- Create a GitHub release:
  - Release name should be a version number following the semver convention
  - Attach the following assets:
    - metadata.yaml
    - infrastructure-components.yaml
    - cluster-template*.yaml

How do I generate Infrastructure-components.yaml?



```
kustomize build config/default/ > infrastructure-components.yaml
# NOTE: replace container image with the one from the registry
```

Community

- Building a provider is just the start
- It's advisable to get involved in the wider CAPI community
  - Attend the office hours calls on Wednesdays
  - Read & comment on issues and enhancement proposals (CAEP)
  - Update your provider when new CAPI versions are released
- To raise awareness or to increase adoption for your new provider
  - Host regular Office Hours
  - Encourage new contributors by having a well-defined README, good first issues
  - Use forums like CAPI Office Hours to talk about your provider
- To donate to kubernetes-sigs
  - Check if your repo follows the kubernetes template project format
  - Fill out the repo migration request
  - Stay on top of the request and answer any queries :)

Wrapping up

There is a lot we haven't covered. Some important areas:

- **Multiple API versions, conversions and Hub/Spoke**

Some resources when you implement your own provider:

- CAPI Repo: https://github.com/kubernetes-sigs/cluster-api
- CAPI Provider Implementers docs: https://cluster-api.sigs.k8s.io/developer/providers/implementers.html
- List of existing providers: **https://cluster-api.sigs.k8s.io/reference/providers.html**
- Kubebuilder docs: https://book.kubebuilder.io/
- Sample Podman provider: https://github.com/capi-samples/cluster-api-provider-podman

Thanks for listening…..any questions?