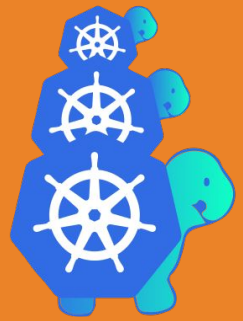
Kubernetes Cluster API



erkan_erol_

About me

- Live in Istanbul/Türkiye
- Platform Engineer @ Giant Swarm
- Netaş -> SAP -> Red Hat
- Kubernetes ~ 5 years
- Kubernetes operators ~ 3 years
- Cluster API ~ 1 year

Erkan Erol

Disclaimer

- This presentation contains some icons from flaticon.com
- This talk contains some copy-paste content from Marcel Müller (twitter.com/MueMarcel)'s old talk.

Scope

For Cluster API users, not for provider maintainers.

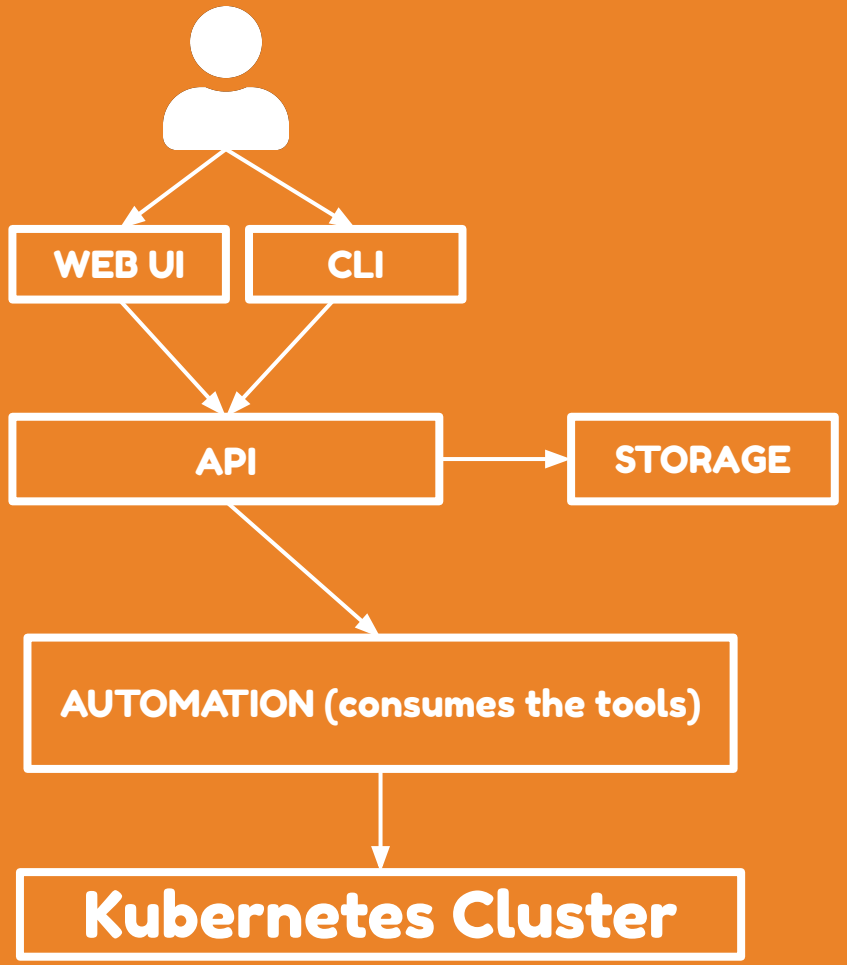
The story

How did we get to this point?

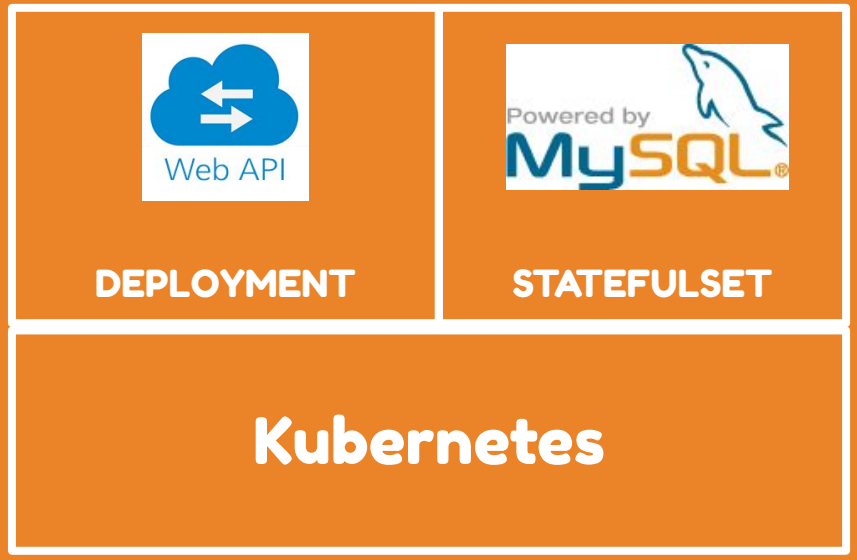
Everything started with its birth...

Installation was not easy. There were many tools.

People were implementing systems to manage lifecycle of Kubernetes clusters.



People started to use k8s to deploy standard workloads by using built-in types.

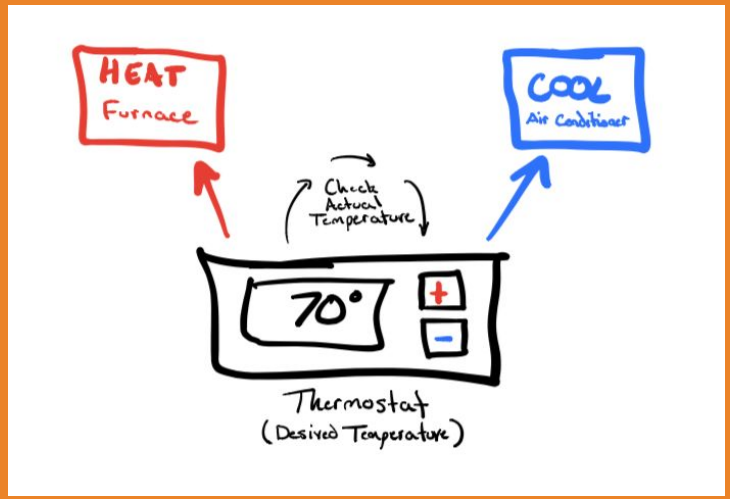


People liked Kubernetes paradigm.

: I desire this state.

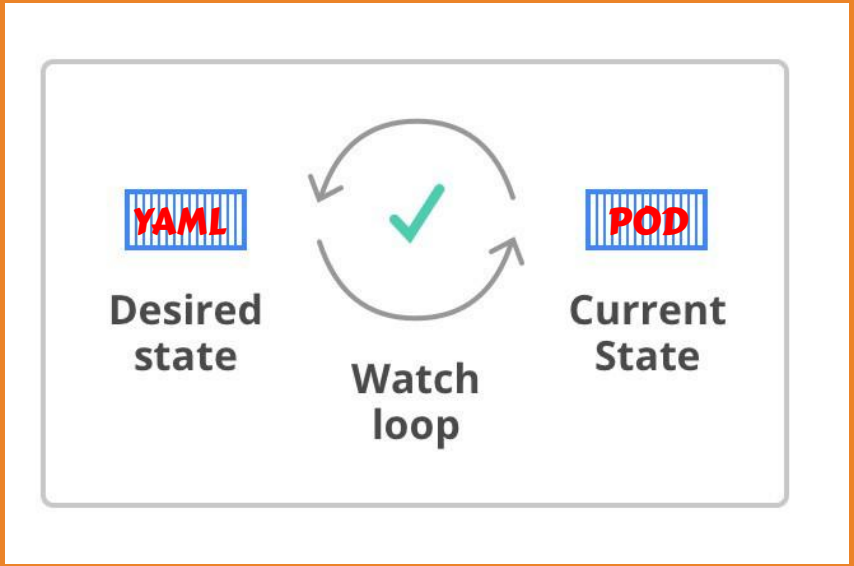
: Here it is.

CONTROLLER PATTERN



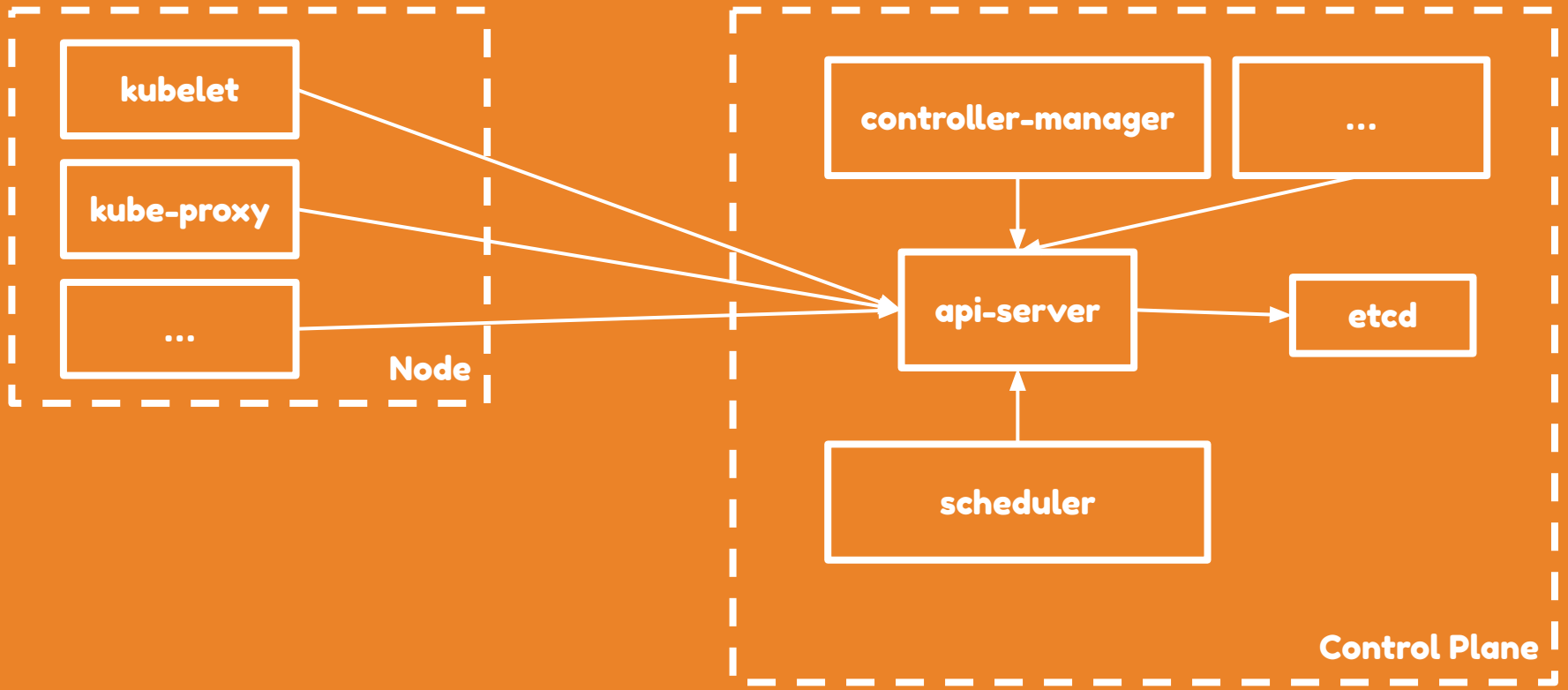
In robotics and automation, a *control loop* is a non-terminating loop that regulates the state of a system.

CONTROLLER PATTERN

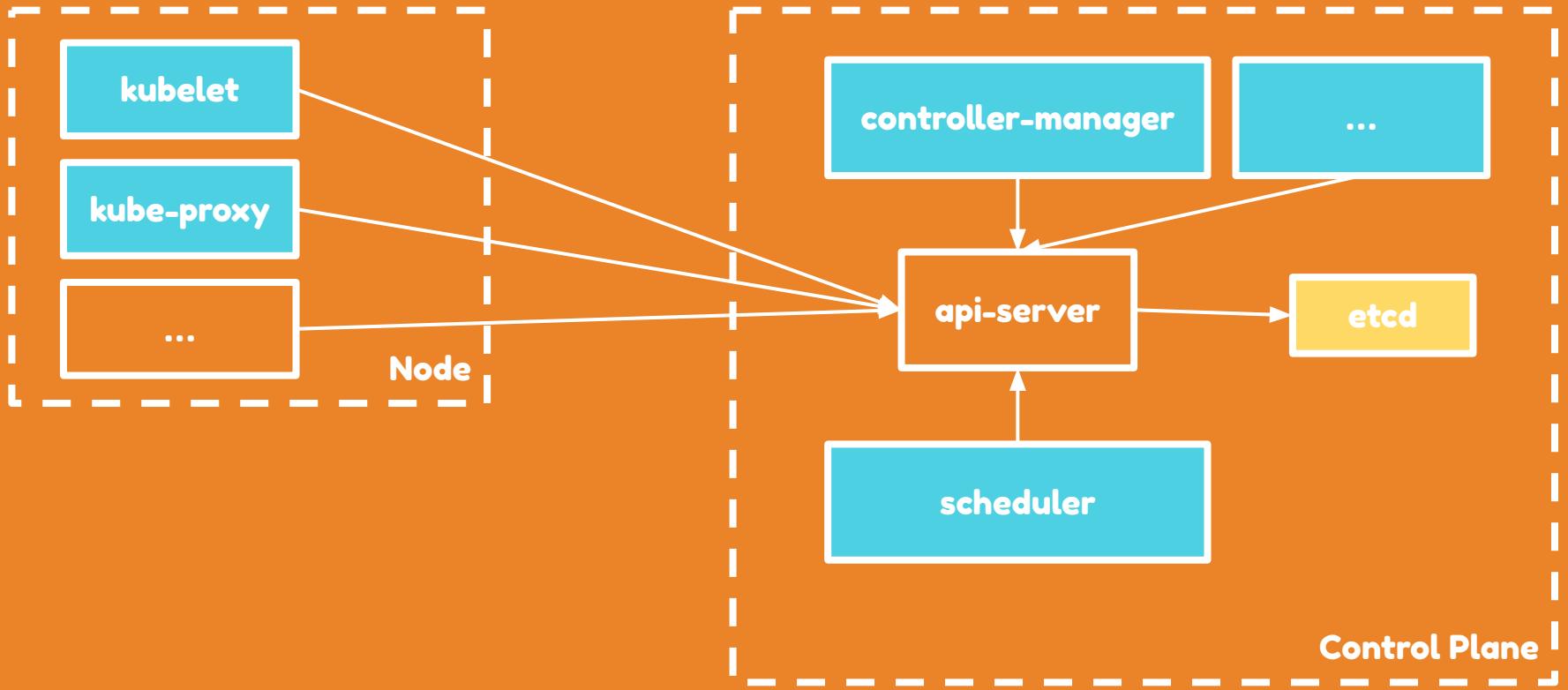


In Kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed.

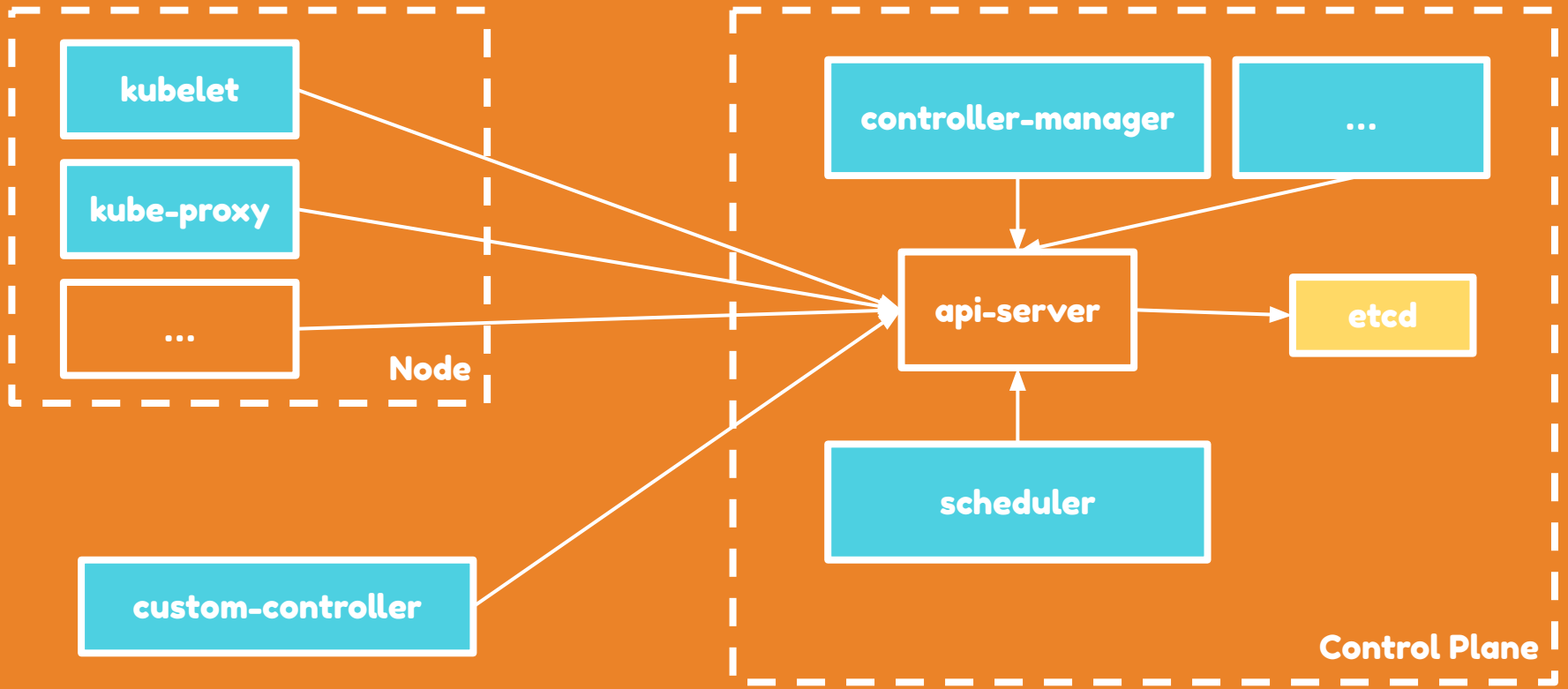
Kubernetes Architecture is actually so simple

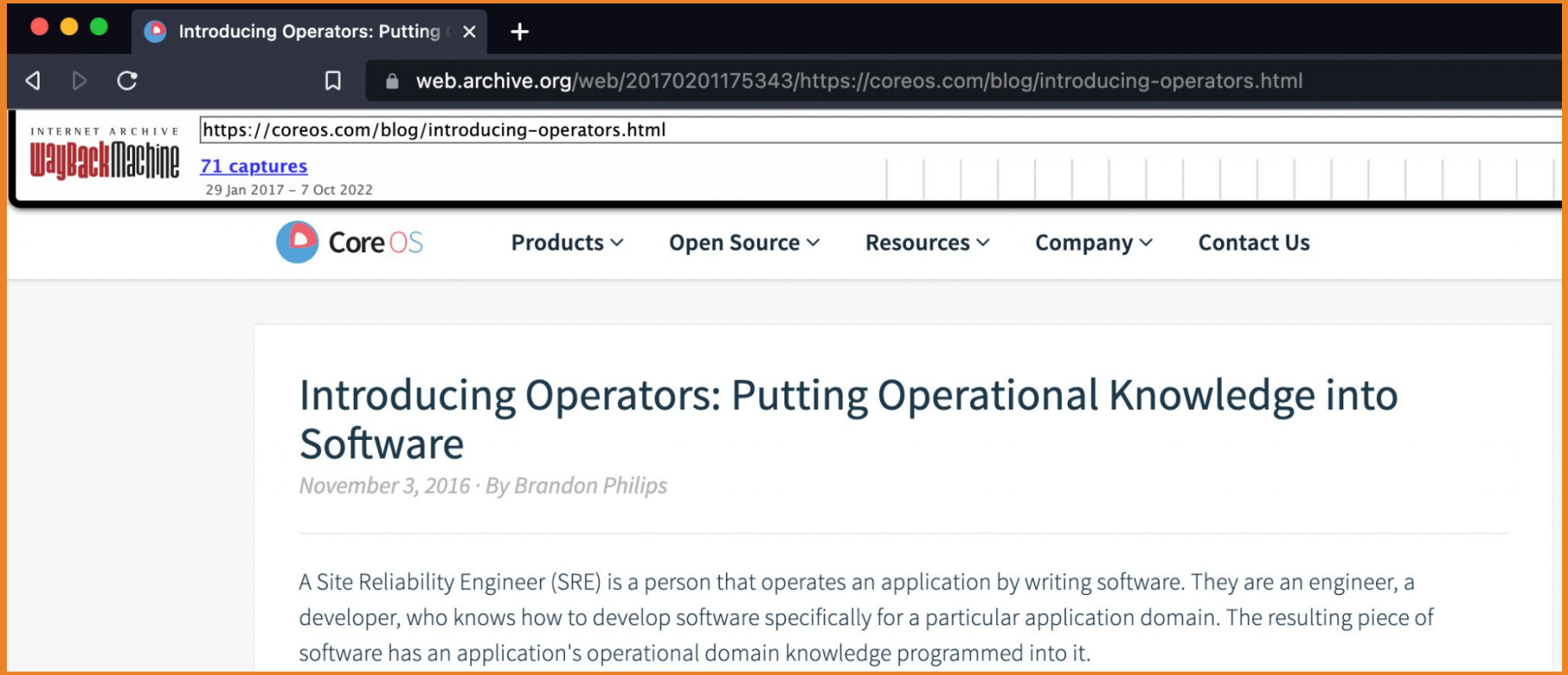


storage(etcd) + common api + controllers



Why don't we add yet another controller?




Introducing Operators: Putting
web.archive.org/web/20170201175343/https://coreos.com/blog/introducing-operators.html
INTERNET ARCHIVE
WayBackMachine
https://coreos.com/blog/introducing-operators.html
71 captures
29 Jan 2017 – 7 Oct 2022
CoreOS
Products
Open Source
Resources
Company
Contact Us
Introducing Operators: Putting Operational Knowledge into Software
November 3, 2016 · By Brandon Philips
A Site Reliability Engineer (SRE) is a person that operates an application by writing software. They are an engineer, a developer, who knows how to develop software specifically for a particular application domain. The resulting piece of software has an application's operational domain knowledge programmed into it.

Operator Pattern

- aims to capture the key aim of a human operator
  - how the system ought to behave
  - how to deploy it
  - how to react if there are problems
- k8s operator = one or more controllers for a specific thing
- e.g. MySql Operator, Prometheus Operator etc.

People ❤️liked❤️ this more

: I desire this

```yaml
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: example
spec:
  replicas: 2
  alerting:
    alertmanagers:
    - namespace: default
      name: alertmanager-example
      port: web
  serviceMonitorSelector:
    matchLabels:
      team: frontend
  ruleSelector:
    matchLabels:
      role: alert-rules
      prometheus: example
```

: Here it is.

prometheus-operator

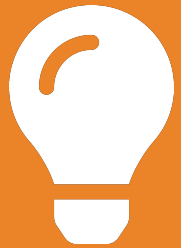
Not only k8s resources but also external resources too.

: I desire this

```yaml
apiVersion: ec2.aws.crossplane.io/v1beta1
kind: VPC
metadata:
  name: sample-vpc
spec:
  forProvider:
    region: us-east-1
    cidrBlock: 10.0.0.0/16
    enableDnsSupport: true
    enableDnsHostNames: true
    instanceTenancy: default
  providerConfigRef:
    name: example
```

: Here it is.

WHY DON'T WE USE OPERATORS TO MANAGE KUBERNETES CLUSTERS?



Giant Swarm

Gardener

KUBERMATIC

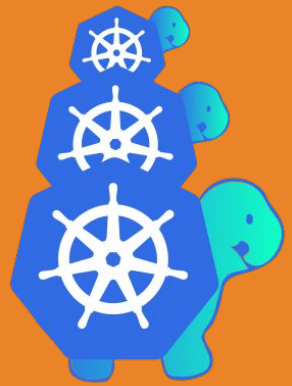
LET'S STANDARDIZE!

Kubernetes Cluster API

Cluster API is a Kubernetes sub-project focused on providing declarative APIs and tooling to simplify provisioning, upgrading, and operating multiple Kubernetes clusters.

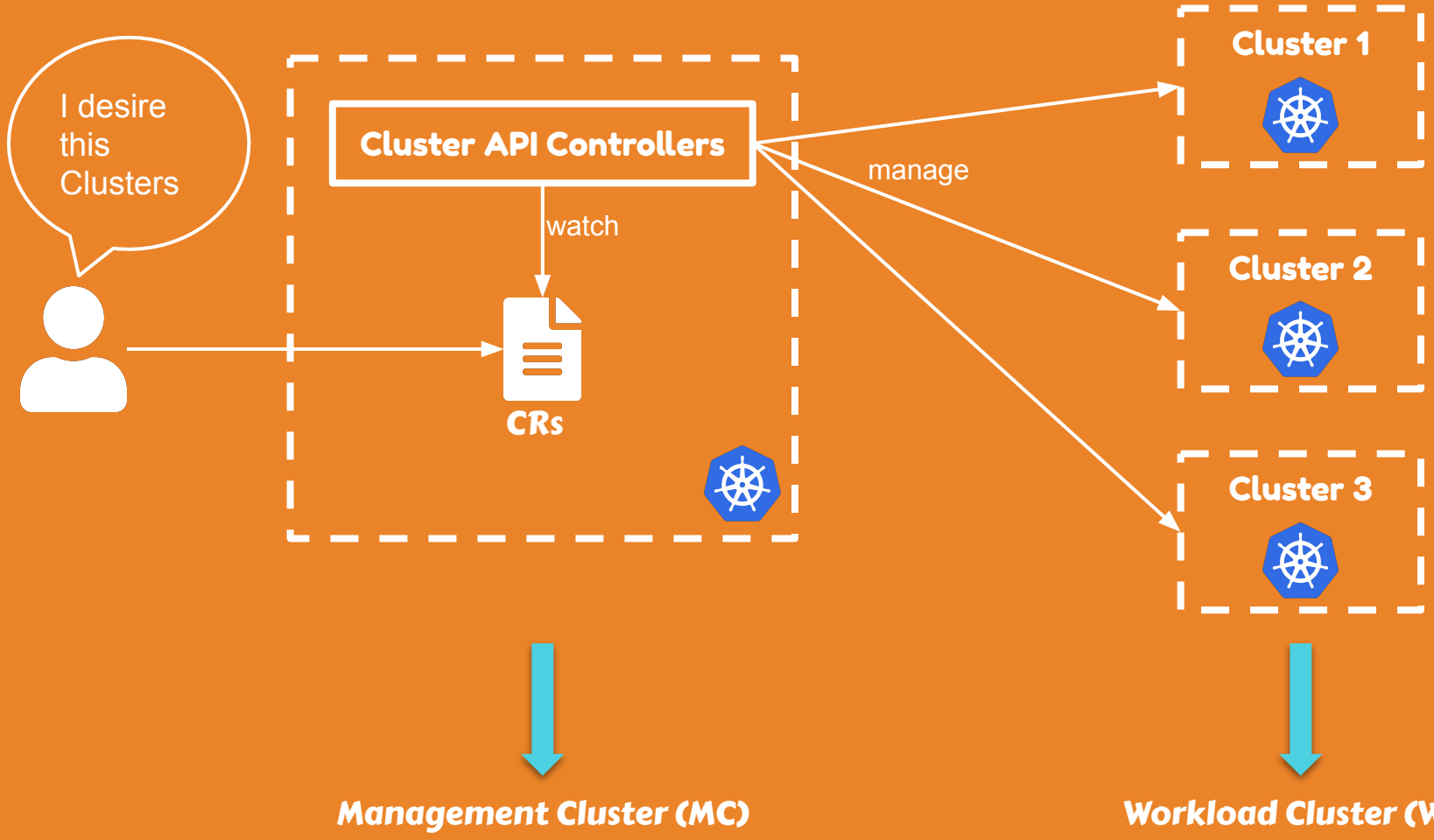
Started by the Kubernetes Special Interest Group (SIG) Cluster Lifecycle, the Cluster API project uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. The supporting infrastructure, like virtual machines, networks, load balancers, and VPCs, as well as the Kubernetes cluster configuration are all defined in the same way that application developers operate deploying and managing their workloads. This enables consistent and repeatable cluster deployments across a wide variety of infrastructure environments.

And it was born



Cluster API Fundamentals

The Tortoise Trainer


I desire this Clusters
Cluster API Controllers
watch
CRs
manage
Cluster 1
Cluster 2
Cluster 3
Management Cluster (MC)
Workload Cluster (WC)

To understand the solution,

Let's think about the problem at first.

What do we need to create a k8s cluster?

I. We need infra resources

Machines / VMs

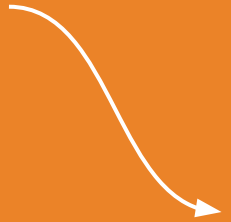
Networking

Storage

Infrastructure Provider

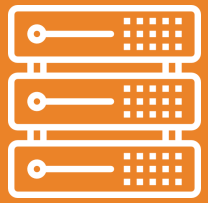
Current Infrastructure Providers

- AWS
- Azure
- Azure Stack HCI
- BYOH
- CloudStack
- DigitalOcean
- Equinix Metal
- GCP
- Hetzner
- Outscale
- IBM Cloud
- 🤍KubeVirt🤍
- MAAS
- Metal3
- Microvm
- Nested
- Nutanix
- OpenStack
- OCI
- Sidero
- Tencent Cloud
- vcluster
- Virtink
- VMware Cloud Director
- vSphere



We will use it as an example
in the upcoming slides

2. We need to convert machines to k8s nodes



- certificates
- joining tokens
- kubelet

Node

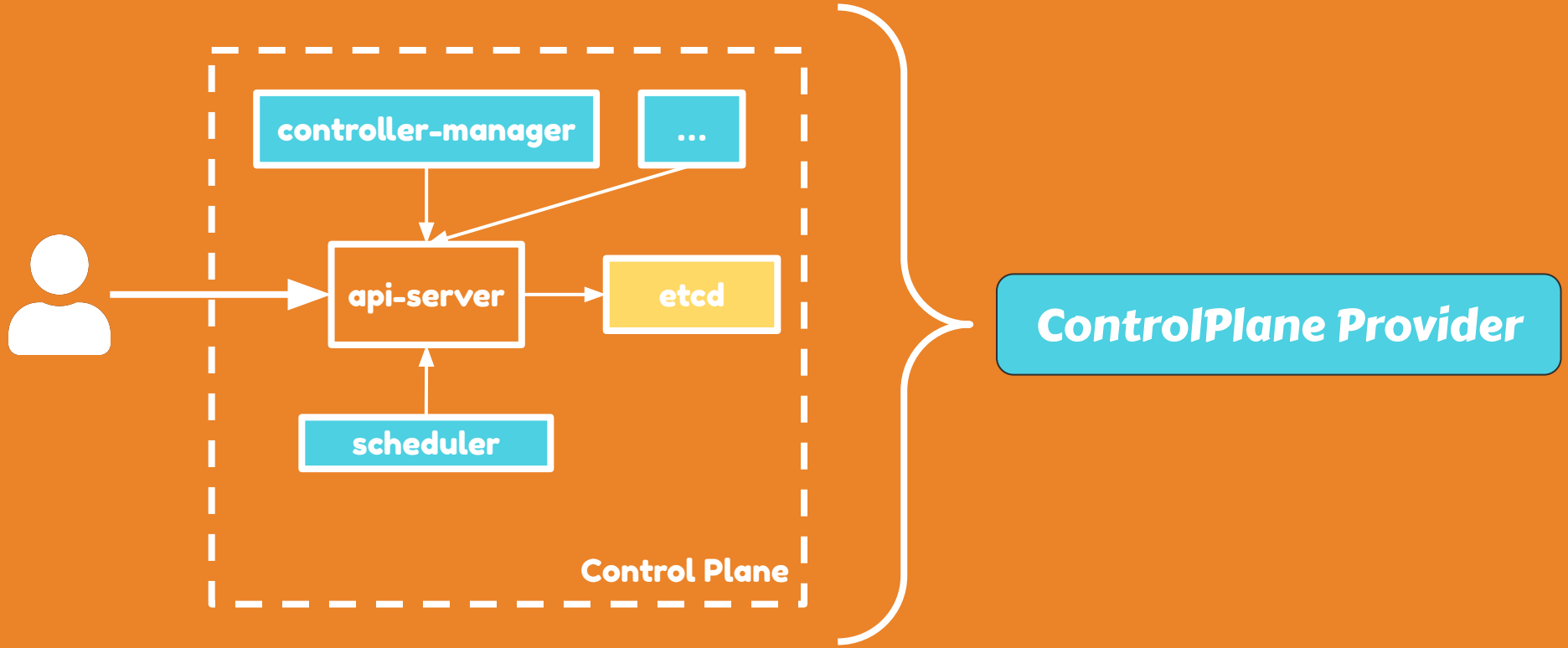
Bootstrap Provider

Current Bootstrap Providers

- EKS
- Kubeadm
- Talos

We will talk about only this one

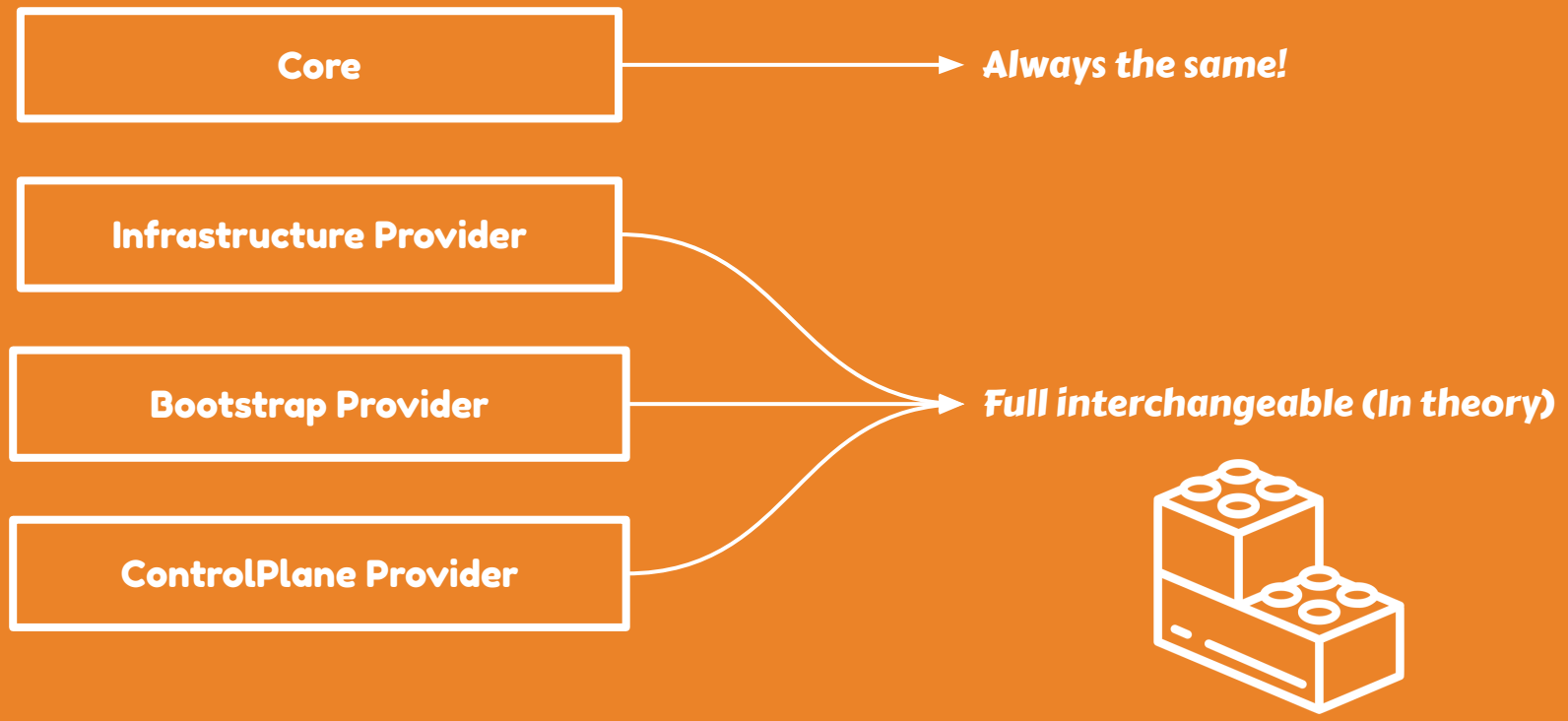
3. We need a control-plane to join our nodes



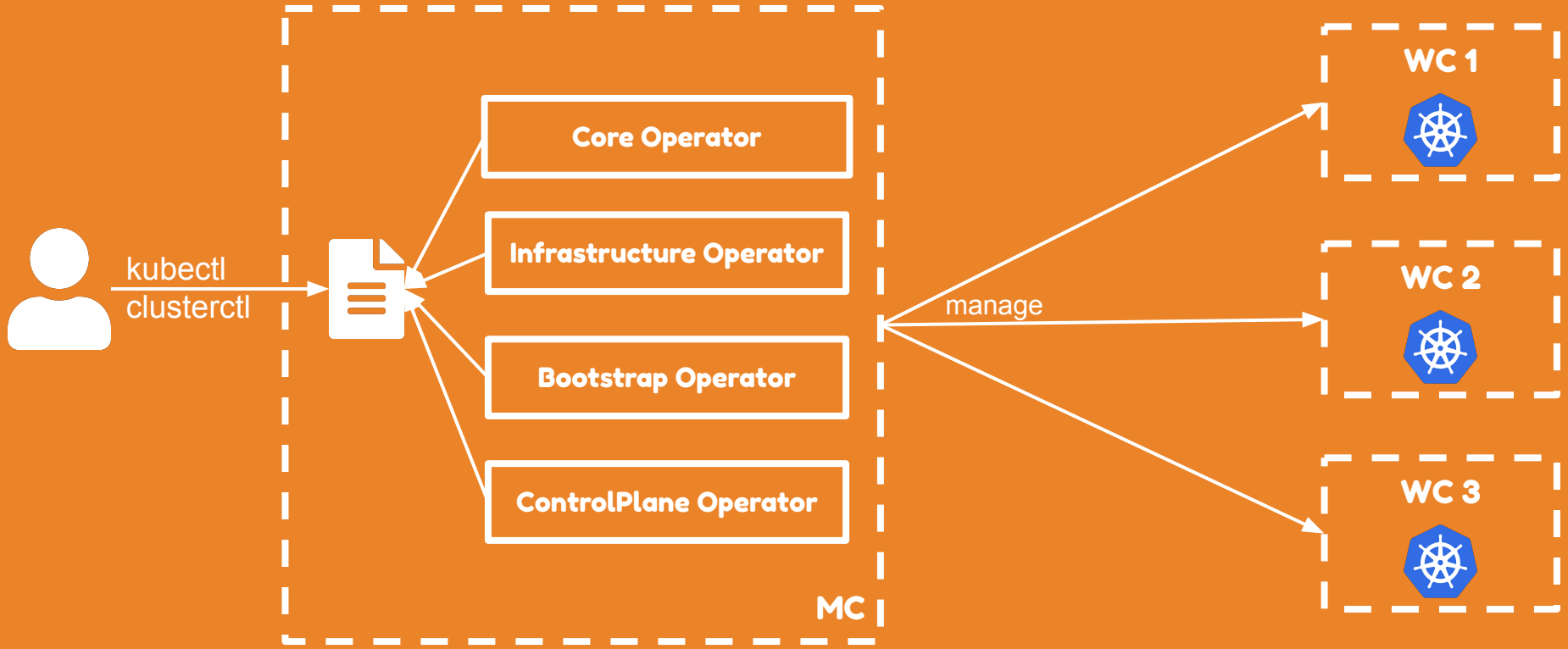
ControlPlane Provider Types

- **Self-provisioned**
  - In WC, controlled by Cluster API
  - *e.g. KubeadmControlPlane* → We will talk about only this one
- **Pod-based**
  - In external cluster, pod based, exposed by a k8s service
- **External or Managed**
  - controlled by another system like GKE, AKS, EKS, etc.

Cluster API components are interchangeable!



BIG PICTURE


kubectl
clusterctl
Core Operator
Infrastructure Operator
Bootstrap Operator
ControlPlane Operator
MC
manage
WC 1
WC 2
WC 3

clusterctl

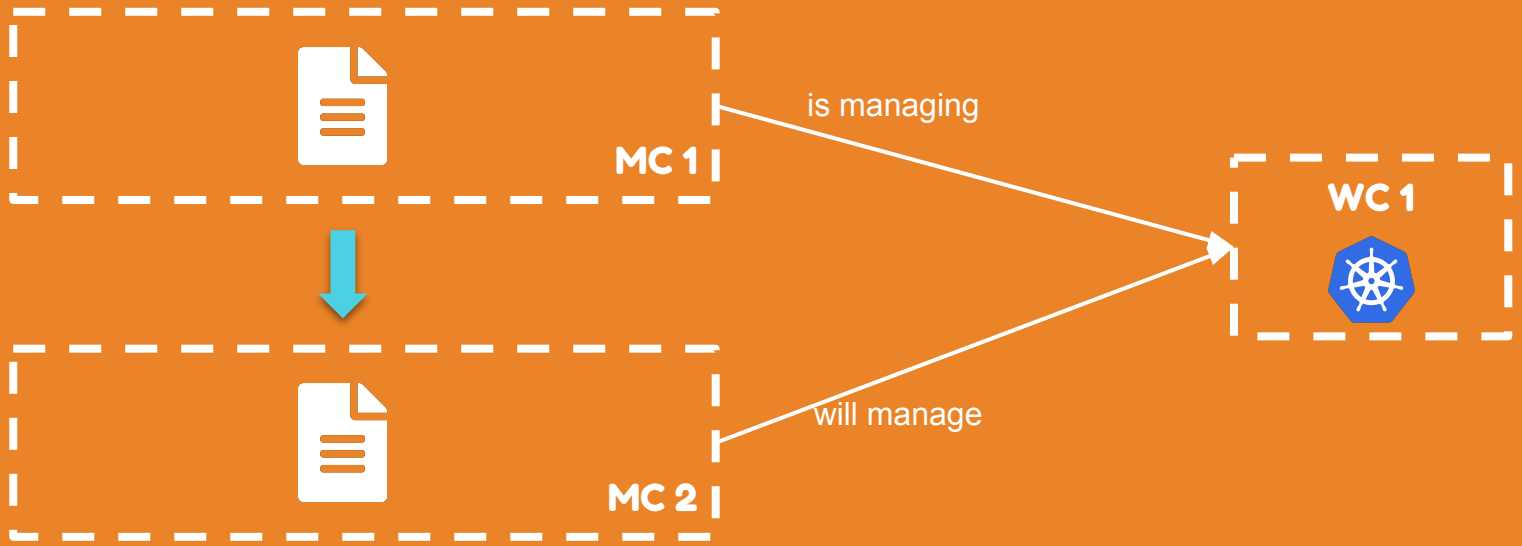
- ***The clusterctl CLI tool handles the lifecycle of a Cluster API management cluster.***
- ***It automates fetching the YAML files defining provider components and installing them.***

- `clusterctl init` Initialize a management cluster.
- `clusterctl upgrade plan` Provide a list of recommended target versions for upgrading Cluster API providers in a management cluster.
- `clusterctl upgrade apply` Apply new versions of Cluster API core and providers in a management cluster.
- `clusterctl delete` Delete one or more providers from the management cluster.
- `clusterctl generate cluster` Generate templates for creating workload clusters.
- `clusterctl generate yaml` Process yaml using clusterctl's yaml processor.
- `clusterctl get kubeconfig` Gets the kubeconfig file for accessing a workload cluster.
- `clusterctl move` Move Cluster API objects and all their dependencies between management clusters.
- `clusterctl alpha rollout` Manages the rollout of Cluster API resources. For example: MachineDeployments.

clusterctl init example

```
→ clusterctl init --infrastructure openstack
Fetching providers
Installing cert-manager Version="v1.7.2"
Waiting for cert-manager to be available...
Installing Provider="cluster-api" Version="v1.2.4" TargetNamespace="capi-system"
Installing Provider="bootstrap-kubeadm" Version="v1.2.4" TargetNamespace="capi-kubeadm-bootstrap-system"
Installing Provider="control-plane-kubeadm" Version="v1.2.4" TargetNamespace="capi-kubeadm-control-plane-system"
Installing Provider="infrastructure-openstack" Version="v0.6.3" TargetNamespace="capo-system"

Your management cluster has been initialized successfully!

You can now create your first workload cluster by running the following:

  clusterctl generate cluster [name] --kubernetes-version [version] | kubectl apply -f -
```

clusterctl move



DREAM vs REALITY

November 13th, 2021

Erkan Erol 7:29 PM

Hi all. I have a high level understanding about cluster-api and I am trying to understand the rationale behind cluster-api architecture/design. I didn't get something.

Before jumping into details, I was under such an impression:

- There are some common types coming from cluster-api core like Cluster, ControlPlane, Machine etc.
- End users are responsible for creating these common types. They have options to pick a specific provider.
- Controllers of providers are watching these common types and they create their own types accordingly and provide the necessary infra.

like the mechanism between PersistentVolumeClaim, StorageClass, PersistentVolume etc.

After checking the examples, my understanding is that users have to create the resources for a specific type like `Metal3Cluster` and put a reference in the `Cluster` obj.

My question: Why? Why don't we have common specs for all providers? Is there a resource about it so that I can check it? (edited)

4 replies Last reply 11 months ago

DREAM vs REALITY

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-spec-in-my-dream
spec:
  infra: OpenStack
  bootstrap: Kubeadm
  controlPlane: GKE
  myGenericSpec:
    workerNodeReplicas: 3
    controlPlaneReplicas: 3
```

There is no one or few common CRs for all providers.

```
→ helm get manifest demo1-cluster |grep '^kind' -B 1
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
--
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
kind: OpenStackCluster
--
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
--
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
--
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
--
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
kind: OpenStackMachineTemplate
--
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineHealthCheck
--
apiVersion: addons.cluster.x-k8s.io/v1beta1
kind: ClusterResourceSet
```

Each provider has its own CRs.

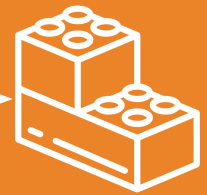
Deep Dive




Deep Dive Stack

- Core Operator
- Openstack as infra provider
- Kubeadm as bootstrap provider
- KubeadmControlPlane as controlplane provider

These are just examples to understand the concepts.
Don't forget



I want to create a cluster with Cluster API.

How can I do it?

kind: Cluster

- Managed by “capi-controller-manager” (Core Operator)
- Main CR
- Give reference to infra provider
- Give reference to controlplane provider (Optional)

kind: Cluster : Spec

As as Cluster API user

I want to have a cluster

With this control plane provider

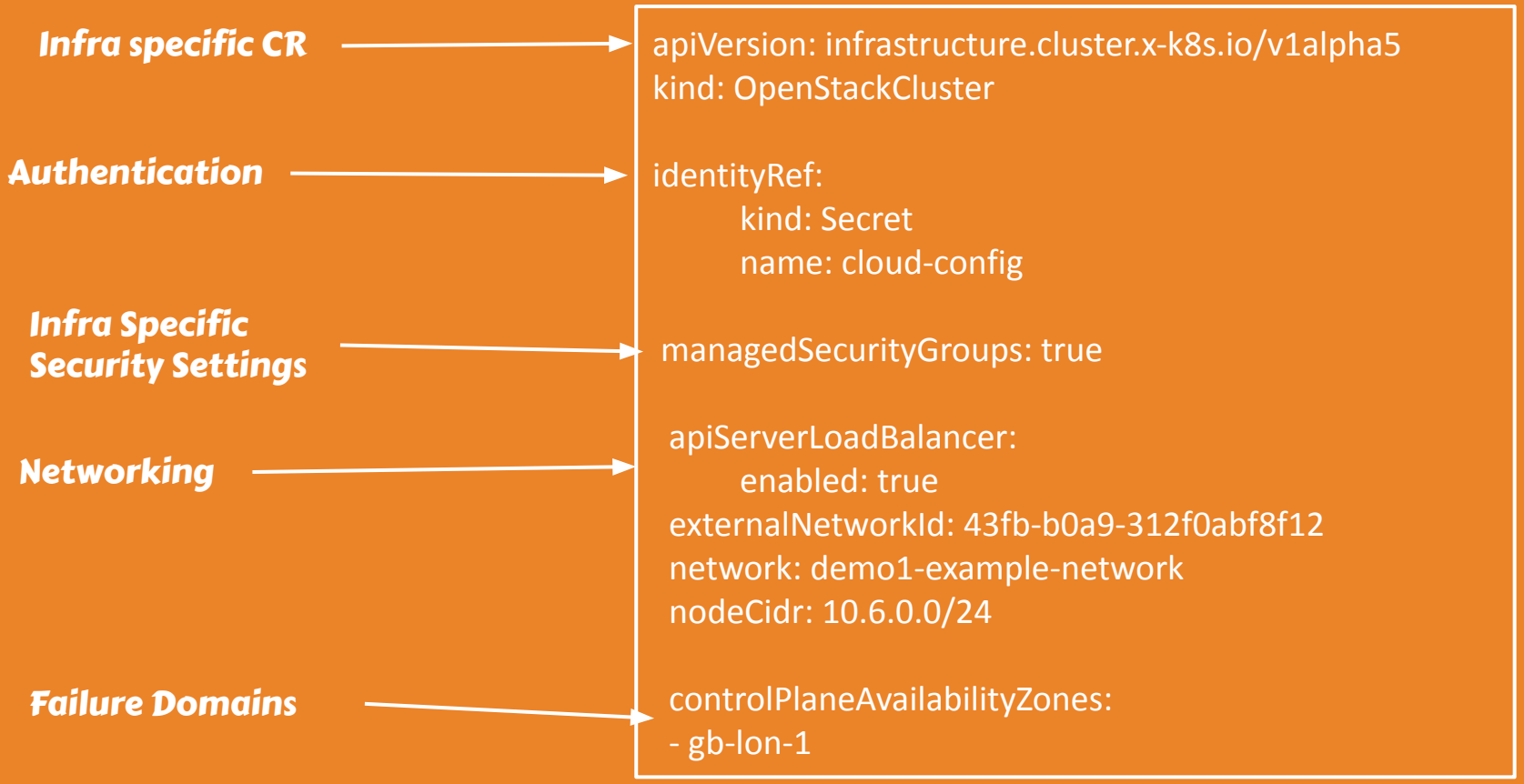
On this infra provider

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster

spec:
 …
  controlPlaneRef:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: KubeadmControlPlane
        name: demo1
        namespace: org-multi-project

  infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
        kind: OpenStackCluster
        name: demo1
        namespace: org-multi-project
```

kind: InfraCluster

- e.g. OpenStackCluster
- Managed by “capo-controller-manager” (Infra Operator)
- CR to declare common/primary infra resources for the cluster
  - Project / Account in infra provider
  - Credentials to access infra provider
  - VPC / Network configuration
  - Region / Availability Zone / Failure Domains

kind: InfraCluster : Spec



With Cluster + InfraCluster, we created some basics like project, vpc, security groups, LB for k8s api etc. but we still don't have any machine!

kind: Machine

- Managed by “capi-controller-manager” (Core Operator)
- Give reference to infra provider
- Give reference to bootstrap provider (Optional)

kind: Machine : Spec

As as Cluster API user

I want to have a machine

On this infra provider

With this bootstrap configuration

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Machine

spec:
   infrastructureRef:
         apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
         kind: OpenStackMachine
         name: demo1-default-91da4cd9-m7gn2
         namespace: org-multi-project
         uid: d7ac11dd-d41c-40a4-9a4b-ab2fe5dd71ad

  bootstrap:
    configRef:
         apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
         kind: KubeadmConfig
         name: demo1-region1-8342e0a3-lwlqx
         namespace: org-multi-project
         uid: 5a31ddc4-92e3-4bc9-9af3-d7827e6366fc
      dataSecretName: demo1-region1-8342e0a3-lwlqx
```

kind: InfraMachine

- e.g. OpenStackMachine
- Managed by “capo-controller-manager” (Infra Operator)
- CR to declare machine spec
  - Image
  - Size / Flavor
  - Network / Subnet
  - Volumes
  - ...

kind: InfraMachine : Spec

Infra specific CR →

Authentication →

Machine Spec →

```
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
kind: OpenStackMachine

spec:
  identityRef:
        kind: Secret
        name: cloud-config

  flavor: n1.medium
  image: ubuntu-2004-kube-v1.22.8
  rootVolume:
        diskSize: 60
```

OK. We have a machine now. Let's talk about how to bootstrap it.

Kind: KubeadmConfig

- Managed by “capi-kubeadm-bootstrap-controller-manager” (Bootstrap Operator)
- CR to declare
  - Bootstrap configuration like kubeadm config
  - Files to mount
  - Users to create
  - Commands to run

kind: KubeadmConfig : Spec

As as Cluster API user

I want to bootstrap my machine

By injecting these files

By using this join configuration

By using this kubelet configuration

By running these commands

By using this format

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfig

spec:
  files:
  - content: |
      ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AA...
    path: /etc/ssh/trusted-user-ca-keys.pem
    permissions: "0600"
  joinConfiguration:
    discovery:
      bootstrapToken:
          apiServerEndpoint: xxx.yyy.ttt.zzz:6443
          token: u7zw64.mytoken
      nodeRegistration:
        kubeletExtraArgs:
          eviction-hard: memory.available<200Mi
          feature-gates: ExpandPersistentVolumes=true
        name: '{{ local_hostname }}'
  postKubeadmCommands:
  - systemctl restart sshd

format: cloud-config
```

How the bootstrapping works - I

Bootstrap operator creates a secret based on the spec in the *KubeAdmConfig* CR.

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfig
status:
  dataSecretName: demo1-region1-8342e0a3-lwlqx
  ready: true
```

How the bootstrapping works - 2

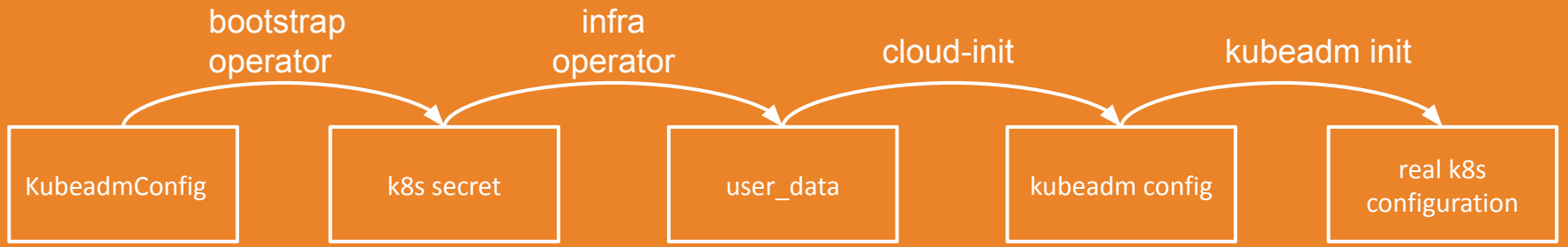
The secret content is in one of the supported formats.

cloud-config files are special scripts designed to be run by the cloud-init service.

```
## template: jinja
#cloud-config

write_files:
-   path: /etc/ssh/trusted-user-ca-keys.pem
            permissions: '0600'
            content: |
            ssh-ed25519 dfdsfs
vault-ca@vault.operations.giantswarm.io

-   path: /run/kubeadm/kubeadm-join-config.yaml
            owner: root:root
            permissions: '0640'
            content: |
            ---
            apiVersion: kubeadm.k8s.io/v1beta3

runcmd:
  - kubeadm join --config
/run/kubeadm/kubeadm-join-config.yaml
  - "systemctl restart sshd"
users:
  - name: giantswarm
            sudo: ALL=(ALL) NOPASSWD:ALL
```

How the bootstrapping works - 3

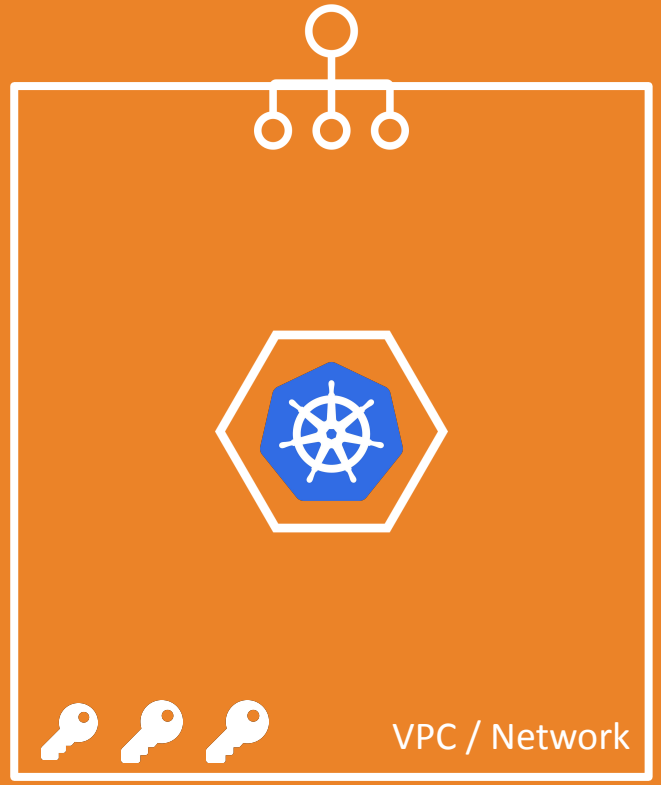
Infra operator fetches bootstrap configuration from Machine CR (by following owner references) and uses that as user_data for cloud-init while creating a machine.



We converted the machine to a k8s node, which is great!

We plan to create more.

Is there a way to manage them as a set instead of one by one?



Cluster API follows k8s approach!


Deployment


Machine Deployment


Replica Set


Machine Set


Pod


Machine

kind: MachineDeployment

- Managed by “capi-controller-manager” (Core Operator)
- Higher level CR to manage Machines like Deployment in Kubernetes
- Refers update strategies and update configuration

kind: MachineDeployment : Spec

part of core api

like k8s deployment

update strategy

machine spec

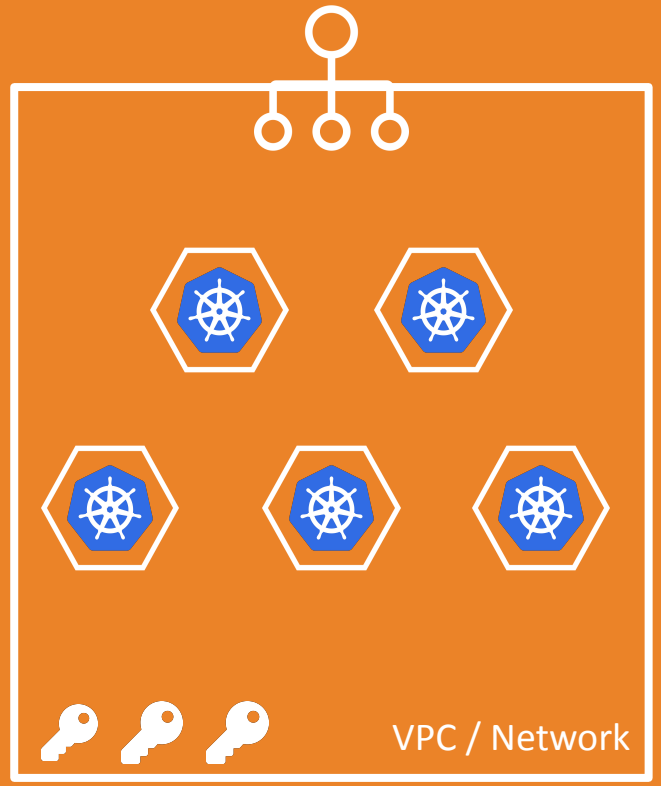
```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment

spec:
  minReadySeconds: 0
  progressDeadlineSeconds: 600

  replicas: 5
  selector:
        matchLabels:
        cluster.x-k8s.io/cluster-name: demo1
        cluster.x-k8s.io/deployment-name: demo1-region1

  strategy:
    rollingUpdate:
        maxSurge: 1
        maxUnavailable: 0
        type: RollingUpdate

  template:
    MACHINE_TEMPLATE
```

Wait! To be able to create all these nodes, we need a control plane to join.

Where is the control plane?

In the beginning, in the Cluster CR, we mentioned *KubeadmControlPlane*.

Let's talk about it.



kind: KubeadmControlPlane

- Managed by “capi-kubeadm-control-plane-controller-manager” (ControlPlane Operator)
- Referred by *Cluster*
- Refers control-plane configuration
  - api-server
  - controller-manager
  - scheduler
  - etcd
- Contains fields of KubeadmConfig too since every control plane node is also a node :)

kind: KubeadmControlPlane : Spec - I

Control-plane configuration

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane

spec:
  kubeadmConfigSpec:
    clusterConfiguration:
      apiServer:
        extraArgs:
          enable-admission-plugins: NamespaceLifecycle,LimitRanger...
      controllerManager:
        extraArgs:
          authorization-always-allow-paths: /healthz,/readyz,/livez,/metrics
          bind-address: 0.0.0.0
      etcd:
        local:
          extraArgs:
            listen-metrics-urls: http://0.0.0.0:2381
            imageRepository: giantswarm
            imageTag: 3.5.4-0-k8s
      scheduler:
```

kind: KubeadmControlPlane : Spec - 2

Like KubeadmConfig since all control-plane nodes are also nodes!

Refers InfraMachineTemplate CR

Like MachineDeployment but for control-plane machines

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
spec:
  kubeadmConfigSpec:
    files:
      format: cloud-config
    initConfiguration:
    joinConfiguration:
    preKubeadmCommands:
  machineTemplate:
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
      kind: OpenStackMachineTemplate
      name: demo1-control-plane-91da4cd9
      namespace: org-multi-project
  replicas: 3
  rolloutStrategy:
        rollingUpdate:
        maxSurge: 1
        type: RollingUpdate
```

How control plane provisioning works - I

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: demo1-rrsnf
spec:
kubeadmConfigSpec:
  clusterConfiguration:
     apiServer:
       certSANs:
         - 127.0.0.1
         - localhost
         - api.demo1.test.erkan.io
```

```
kind: Secret
metadata:
  name: demo1-ca
```

```
kind: Secret
metadata:
  name: demo1-etcd
```

```
kind: Secret
metadata:
  name: demo1-sa
```

```
kind: Secret
metadata:
  name: demo1-proxy
```

```
kind: Secret
metadata:
  name: demo1-kubeconfig
```

When cluster infrastructure is ready, ControlPlane operator provisions certificates and kubeconfig for the cluster. It puts those into k8s secrets.

Everyone uses this to access WC.

How control plane provisioning works - 2

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: demo1-rrsnf
spec:
  clusterConfiguration:
    controlPlanePart:
    standardNodeConfigurationPart:

  machineTemplate:
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
      kind: OpenStackMachineTemplate
      name: demo1-control-plane-91da4cd9
```

```
kind: KubeadmConfig
metadata:
  name: demo1-rrsnf
```

ControlPlane operator creates a special *KubeadmConfig* for controlplane nodes by using control-plane specific and generic configuration in *KubeadmControlPlane* CR. This KubeadmConfig includes certificates too.

How control plane provisioning works - 3

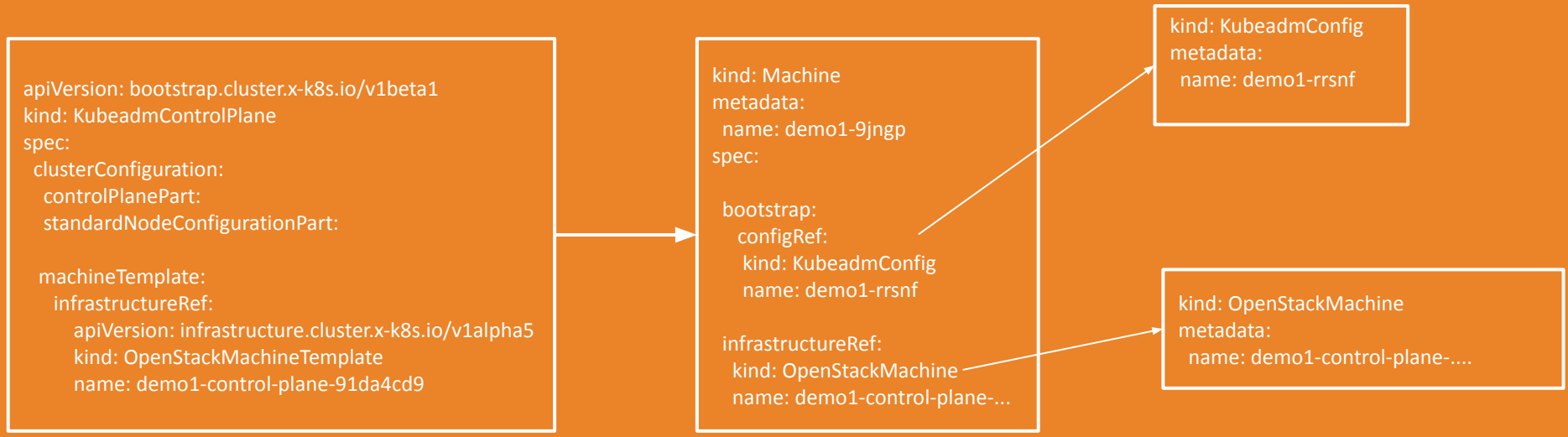
```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
spec:
  clusterConfiguration:
    controlPlanePart:
    standardNodeConfigurationPart:

    machineTemplate:
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1alpha5
        kind: OpenStackMachineTemplate
        name: demo1-control-plane-91da4cd9
```

```
kind: KubeadmConfig
metadata:
  name: demo1-rrsnf
```

```
kind: OpenStackMachine
metadata:
  name: demo1-control-plane-....
```

ControlPlane operator creates a InfraMachine CR by using *InfraMachineTemplate* reference in *KubeadmControlPlane* CR.

How control plane provisioning works - 4

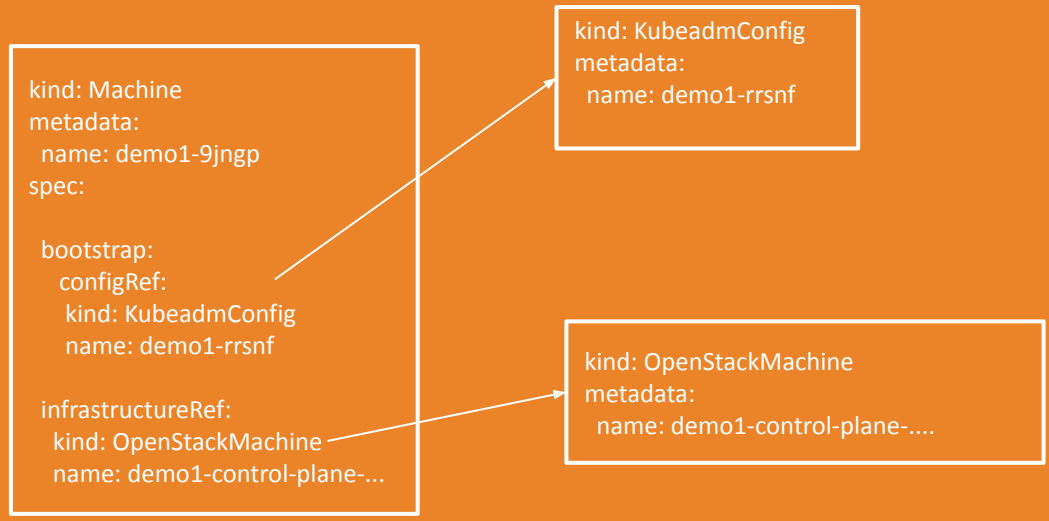


ControlPlane operator finally creates a Machine CR that refers the KubeadmConfig and InfraMachine CRs created earlier.

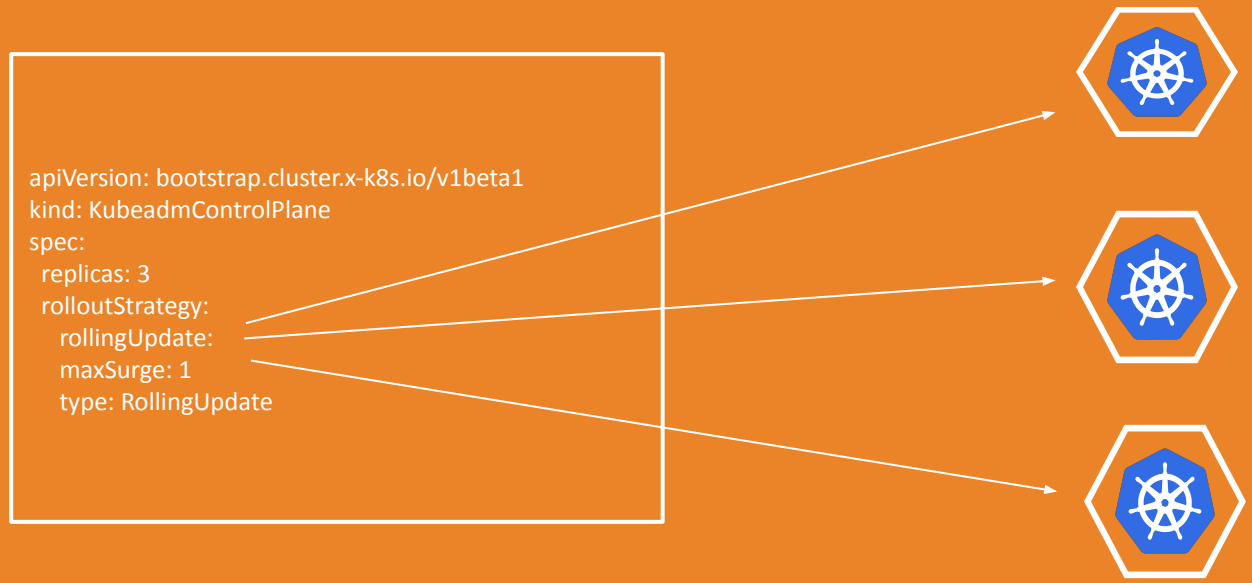
How control plane provisioning works - 5

As we talked earlier, Infra operator creates a machine by using cloud-config provided by Bootstrap operator.

At the end, the created node is a control-plane node.



How control plane provisioning works - 6



ControlPlane operator provisions new control-plane nodes according to "replicas" field and rollout nodes during upgrades according to KubeadmControlPlane spec.

How control plane provisioning works - 7

- ControlPlane operator is not only creating machines like MachineSet.
- It also access & manipulate workload cluster internals to manage control plane nodes in a stable way.

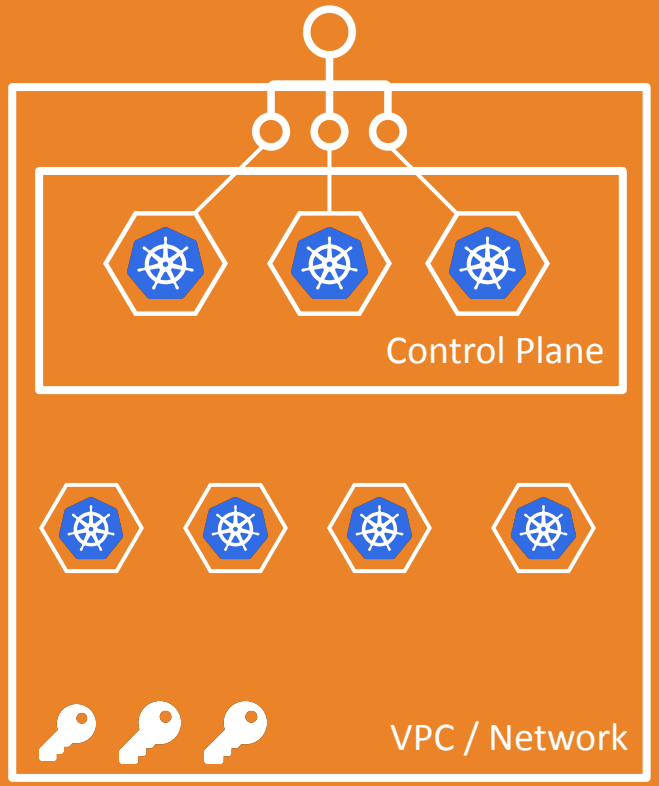
How control plane provisioning works - 8

```go
type WorkloadCluster interface {
	// Basic health and status checks.
	ClusterStatus(ctx context.Context) (ClusterStatus, error)
	UpdateStaticPodConditions(ctx context.Context, controlPlane *ControlPlane)
	UpdateEtcdConditions(ctx context.Context, controlPlane *ControlPlane)
	EtcdMembers(ctx context.Context) ([]string, error)
	GetAPIServerCertificateExpiry(ctx context.Context, kubeadmConfig *bootstrapv1.KubeadmConfig, nodeName string) (*time.Time, error)

	// Upgrade related tasks.
	ReconcileKubeletRBACBinding(ctx context.Context, version semver.Version) error
	ReconcileKubeletRBACRole(ctx context.Context, version semver.Version) error
	UpdateKubernetesVersionInKubeadmConfigMap(ctx context.Context, version semver.Version) error
	UpdateImageRepositoryInKubeadmConfigMap(ctx context.Context, imageRepository string, version semver.Version) error
	UpdateEtcdVersionInKubeadmConfigMap(ctx context.Context, imageRepository, imageTag string, version semver.Version) error
	UpdateEtcdExtraArgsInKubeadmConfigMap(ctx context.Context, extraArgs map[string]string, version semver.Version) error
	UpdateAPIServerInKubeadmConfigMap(ctx context.Context, apiServer bootstrapv1.APIServer, version semver.Version) error
	UpdateControllerManagerInKubeadmConfigMap(ctx context.Context, controllerManager bootstrapv1.ControlPlaneComponent, version semver.Version) error
	UpdateSchedulerInKubeadmConfigMap(ctx context.Context, scheduler bootstrapv1.ControlPlaneComponent, version semver.Version) error
	UpdateKubeletConfigMap(ctx context.Context, version semver.Version) error
	UpdateKubeProxyImageInfo(ctx context.Context, kcp *controlplanev1.KubeadmControlPlane, version semver.Version) error
	UpdateCoreDNS(ctx context.Context, kcp *controlplanev1.KubeadmControlPlane, version semver.Version) error
	RemoveEtcdMemberForMachine(ctx context.Context, machine *clusterv1.Machine) error
	RemoveMachineFromKubeadmConfigMap(ctx context.Context, machine *clusterv1.Machine, version semver.Version) error
	RemoveNodeFromKubeadmConfigMap(ctx context.Context, nodeName string, version semver.Version) error
	ForwardEtcdLeadership(ctx context.Context, machine *clusterv1.Machine, leaderCandidate *clusterv1.Machine) error
	AllowBootstrapTokensToGetNodes(ctx context.Context) error

	// State recovery tasks.
	ReconcileEtcdMembers(ctx context.Context, nodeNames []string, version semver.Version) ([]string, error)
}
```

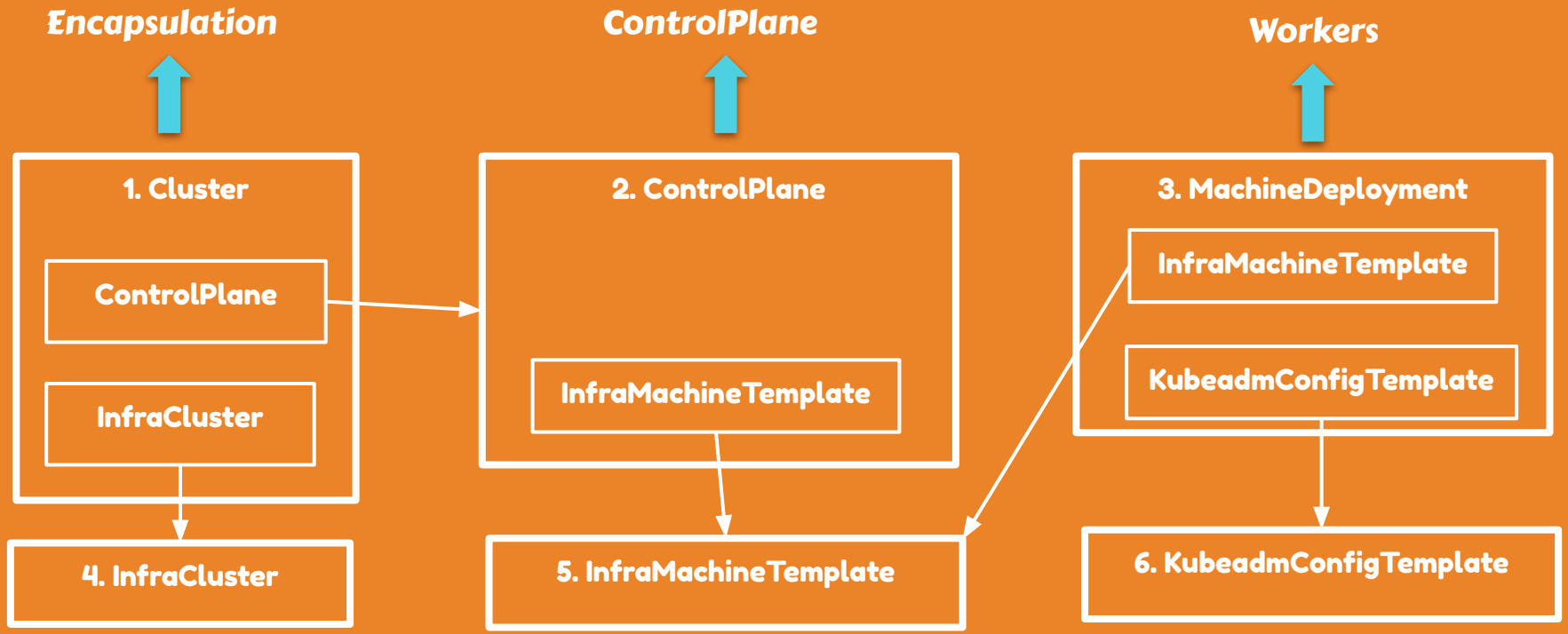
https://github.com/kubernetes-sigs/cluster-api/blob/main/controlplane/kubeadm/internal/workload_cluster.go

We have a fully working cluster now.

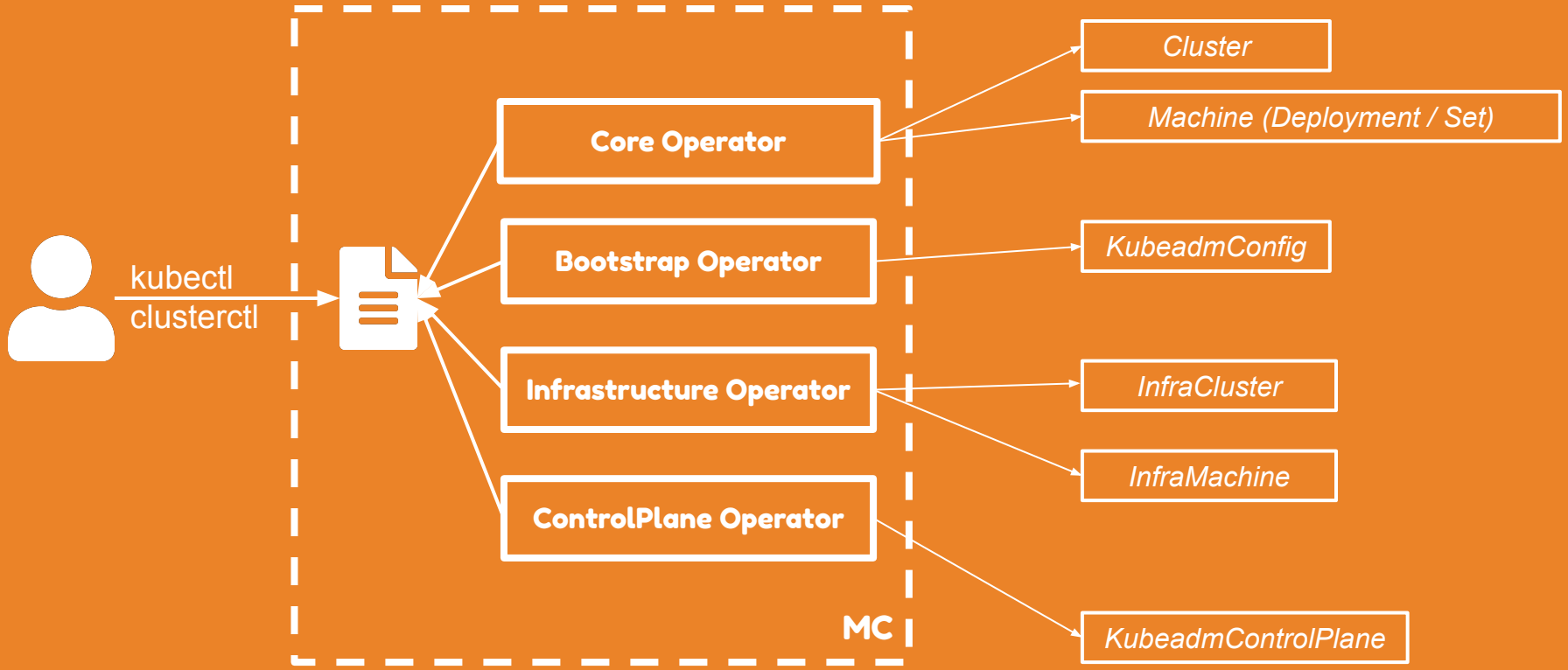
😎😎😎



RECAP-1


Encapsulation
ControlPlane
Workers
1. Cluster
ControlPlane
InfraCluster
2. ControlPlane
InfraMachineTemplate
3. MachineDeployment
InfraMachineTemplate
KubeadmConfigTemplate
4. InfraCluster
5. InfraMachineTemplate
6. KubeadmConfigTemplate

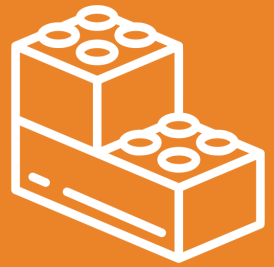
RECAP - 2


kubectl
clusterctl
Core Operator
Bootstrap Operator
Infrastructure Operator
ControlPlane Operator
MC
Cluster
Machine (Deployment / Set)
KubeadmConfig
InfraCluster
InfraMachine
KubeadmControlPlane

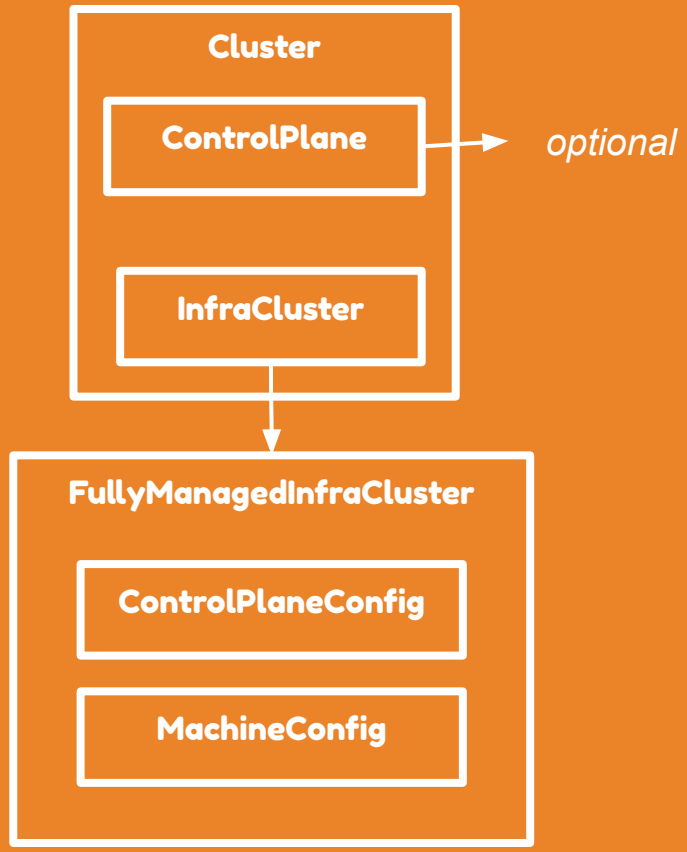
RECAP - 3

```
→ clusterctl describe cluster demo1 --grouping=false --show-machinesets
NAME
Cluster/demo1
├─ClusterInfrastructure - OpenStackCluster/demo1
├─ControlPlane - KubeadmControlPlane/demo1
│ └─Machine/demo1-9jngp
│   └─MachineInfrastructure - OpenStackMachine/demo1-control-plane-91da4cd9-ggbhd
└─Workers
  └─MachineDeployment/demo1-region1
    └─MachineSet/demo1-region1-7776fdbf86
      ├─Machine/demo1-region1-7776fdbf86-d98lv
      │ └─MachineInfrastructure - OpenStackMachine/demo1-default-91da4cd9-6q9km
      ├─Machine/demo1-region1-7776fdbf86-j5r8n
      │ └─MachineInfrastructure - OpenStackMachine/demo1-default-91da4cd9-7hvwc
      └─Machine/demo1-region1-7776fdbf86-t9jz2
        └─MachineInfrastructure - OpenStackMachine/demo1-default-91da4cd9-m7gn2
```

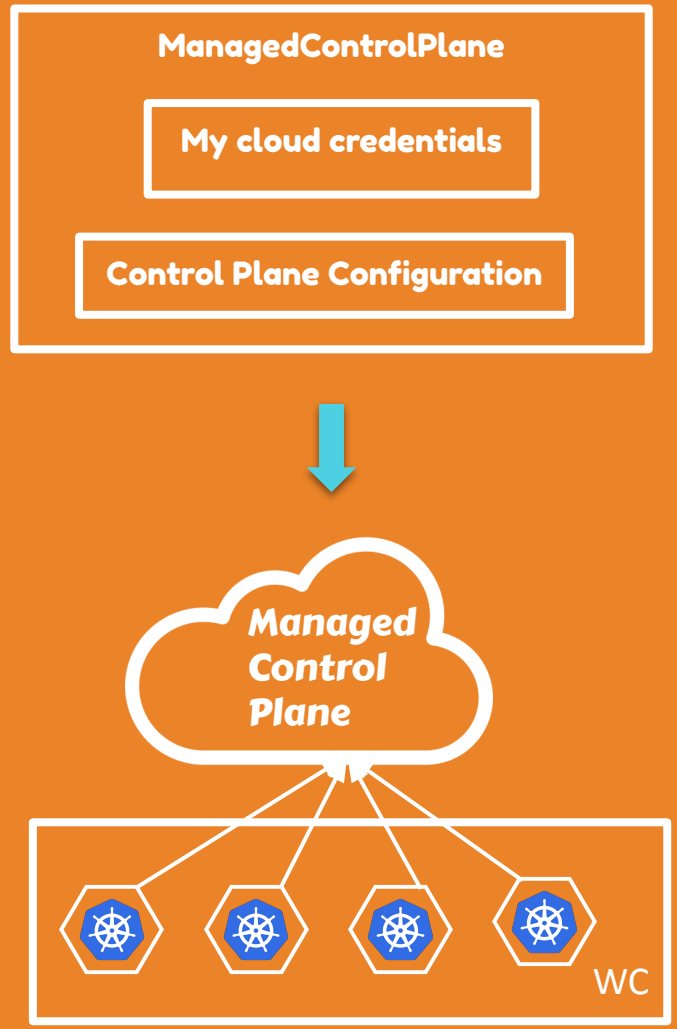
Let's talk about interchangeability again.



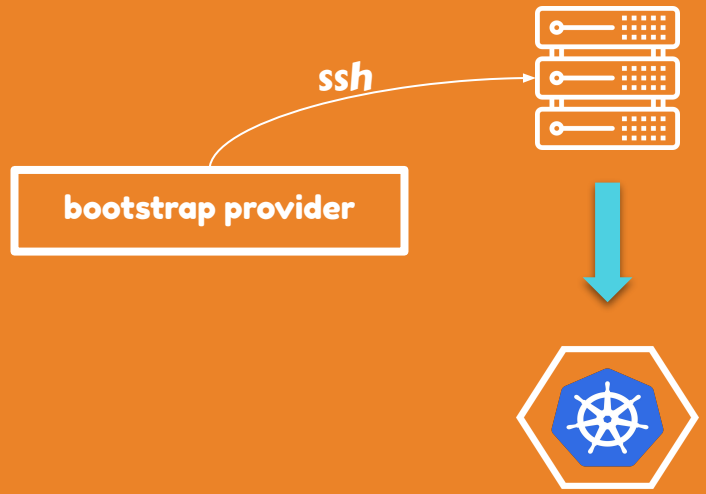
An infra provider can implement only InfraCluster without using others CRs. It is up to infra provider.



A controlplane provider can provision the control plane in a cloud service without using bootstrap/machine apis.



A bootstrap provider can bootstrap a machine by connecting it to through ssh and running commands without using cloud-init. It is up to bootstrap provider.



Cluster API components are interchangeable!

Core

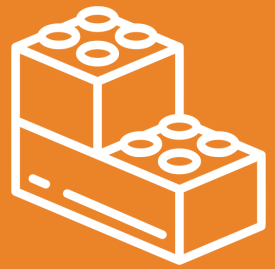
Always the same!

Bootstrap Provider

Infrastructure Provider

ControlPlane Provider

Full interchangeable (In theory)



We don't have so many working examples yet.

BONUS PART

Generated by DALL-E

kind: MachineHealthCheck

- Part of core api.
- Core operator watches machine. If there is a machine which is not in Ready state, it deletes that machine and create a new one.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: demo1
spec:
  clusterName: demo1
  maxUnhealthy: 40%
  nodeStartupTimeout: 20m0s

  selector:
    matchLabels:
      cluster.x-k8s.io/cluster-name: demo1

 unhealthyConditions:
  - status: Unknown
    timeout: 10m0s
    type: Ready
  - status: "False"
    timeout: 10m0s
    type: Ready
```

kind: ClusterResourceSet

As as Cluster API user

In every cluster with these labels

I want to create this k8s object automatically.

```
apiVersion: addons.cluster.x-k8s.io/v1beta1
kind: ClusterResourceSet

metadata:
  name: demo1-coredns

spec:
  clusterSelector:
    matchLabels:
      cluster.x-k8s.io/cluster-name: demo1

  resources:
  - kind: ConfigMap
    name: demo1-coredns
  strategy: ApplyOnce
```

kind: ClusterClass - I

- Experimental feature
- Feature gate name: *ClusterTopology*
- Creating a topology once and using many times

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  name: openstack-clusterclass-v0.1.0
spec:
  controlPlane:
    ref:
      apiVersion: controlplane.cluster.x-k8s.io/v1beta1
      kind: KubeadmControlPlaneTemplate
      name: openstack-clusterclass-v0.1.0
      namespace: default
    machineInfrastructure:
      ref:
        kind: openstackMachineTemplate
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        name: openstack-clusterclass-v0.1.0
        namespace: default
  infrastructure:
    ref:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: OpenstackClusterTemplate
      name: openstack-clusterclass-v0.1.0-control-plane
      namespace: default
  workers:
    machineDeployments:
    - class: default-worker
      template:
        bootstrap:
          ref:
            apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
            kind: KubeadmConfigTemplate
            name: Openstack-clusterclass-v0.1.0-default-worker
            namespace: default
        infrastructure:
          ref:
            apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
            kind: OpenstackMachineTemplate
            name: openstack-clusterclass-v0.1.0-default-worker
            namespace: default
```

kind: ClusterClass - 2

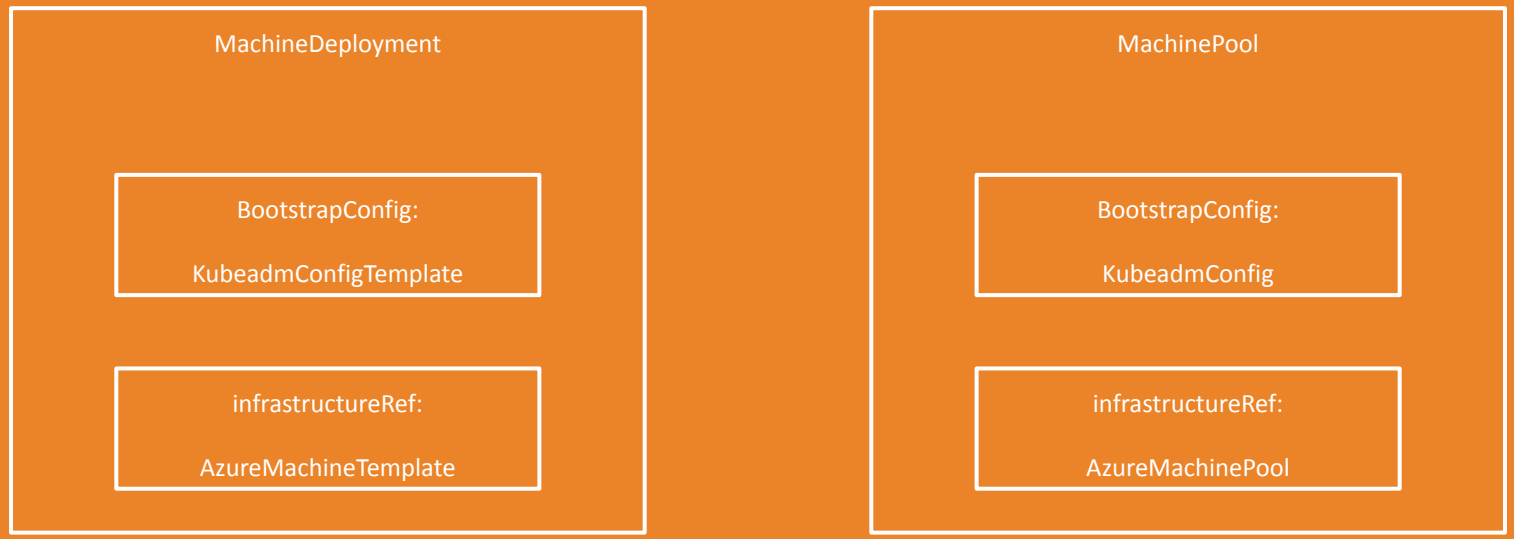
- *Experimental feature*
- *Feature gate name: ClusterTopology*
- *Creating a topology once and using many times*

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: my-openstack-cluster
spec:
  topology:
    class: openstack-clusterclass
    version: v1.22.4
    controlPlane:
      replicas: 3
      metadata:
        labels:
          cpLabel: cpLabelValue
        annotations:
          cpAnnotation: cpAnnotationValue
    workers:
      machineDeployments:
      - class: default-worker
        name: md-0
        replicas: 4
        metadata:
          labels:
            mdLabel: mdLabelValue
          annotations:
            mdAnnotation: mdAnnotationValue
        failureDomain: region
```

kind: MachinePool - 1

- Experimental feature
- Feature gate name: MachinePool
- Like MachineDeployment but doesn't use MachineSet
- Delegates the responsibility of these concerns to an infrastructure provider specific resource such as AWS Auto Scale Groups, GCP Managed Instance Groups, and Azure Virtual Machine Scale Sets.

kind: MachinePool - 2



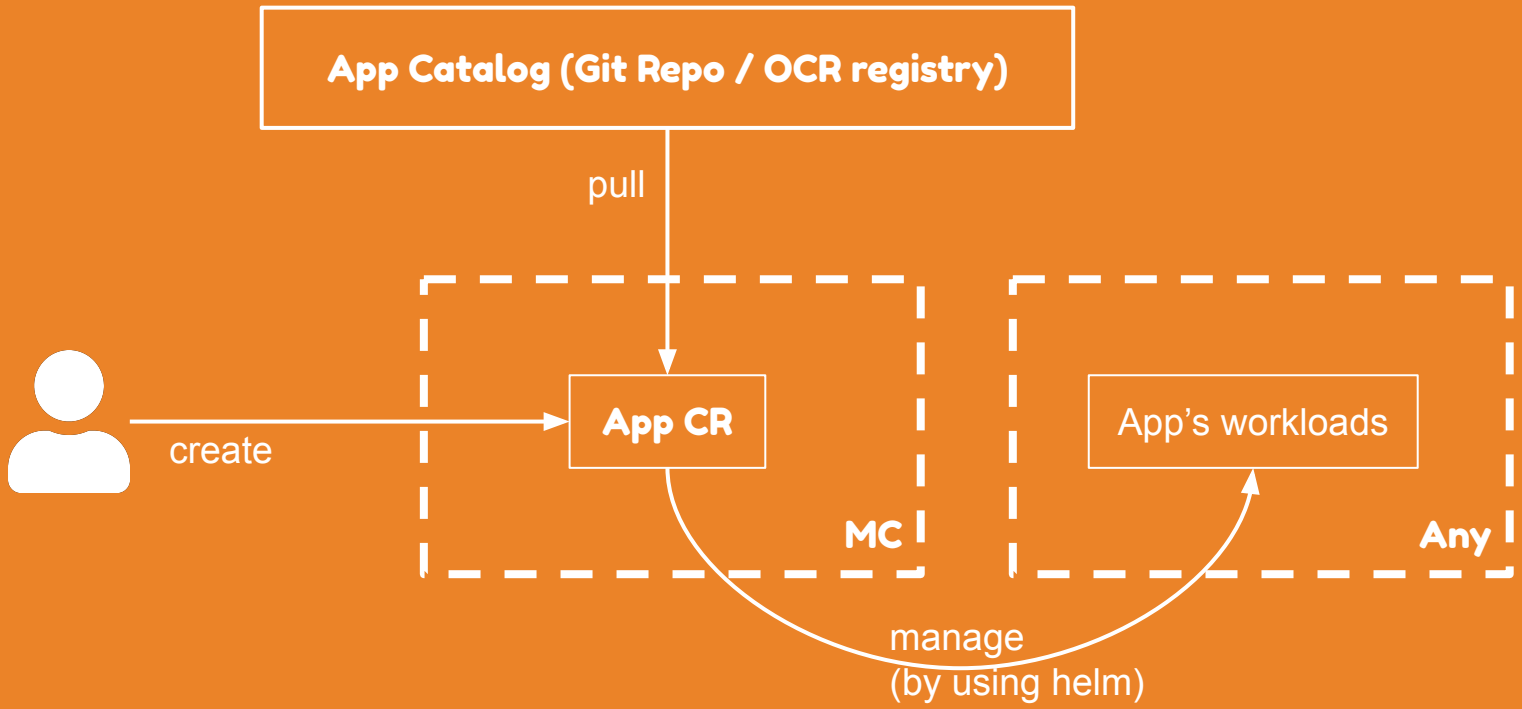
Let me talk about my company for 5 minutes

But not like a salesman. It will be like tips and tricks.

Sauce of Giant Swarm

PUBLIC

App Platform - I



https://docs.giantswarm.io/app-platform

App Platform - 2

- There are lots of apps in the app catalogs
  - giantswarm/giantswarm-catalog
  - giantswarm/default-catalog
- We use the app platform to create WCs too.
  - giantswarm/cluster-api-app
  - giantswarm/cluster-catalog
- Upgrade = changing app version

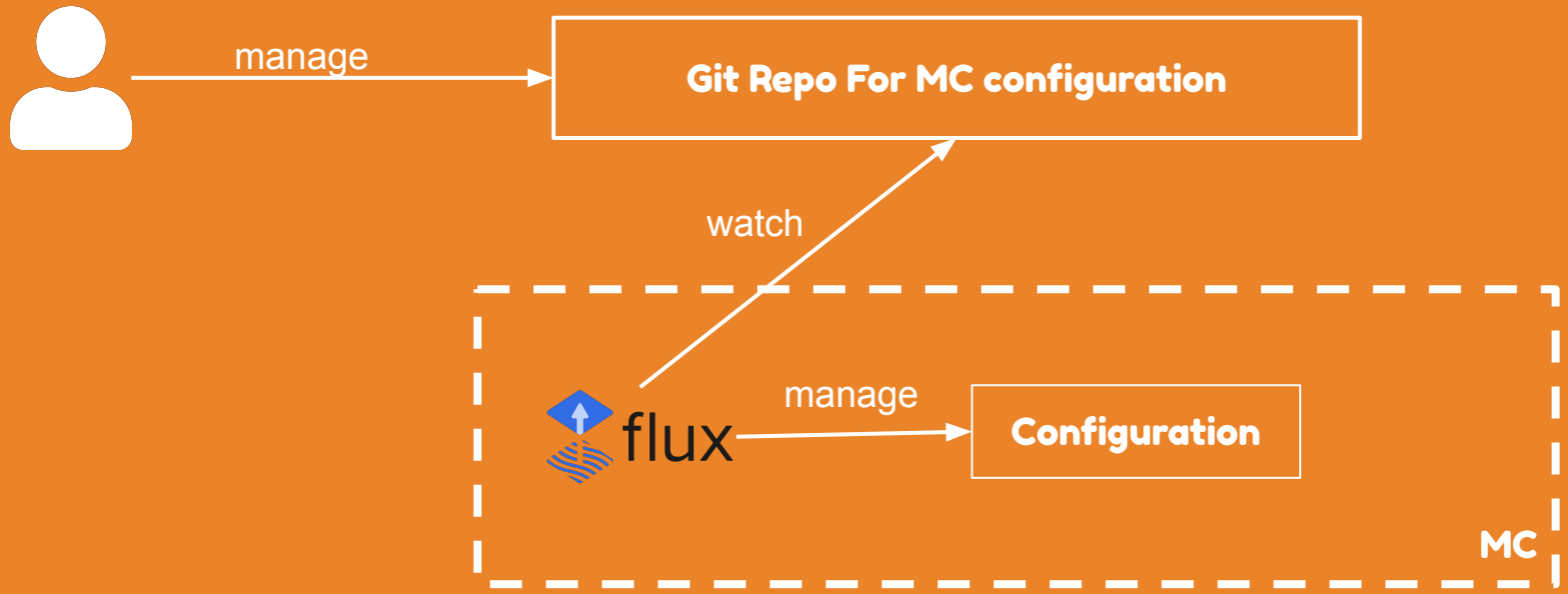
```yaml
apiVersion: application.giantswarm.io/v1alpha1
kind: App
metadata:
  name: demo1-cluster
  namespace: org-multi-project
spec:
  catalog: cluster
  name: cluster-openstack
  namespace: org-multi-project
  userConfig:
    configMap:
      name: demo1-cluster-userconfig
      namespace: org-multi-project
  version: 0.16.1
```

https://docs.giantswarm.io/app-platform

Example Apps

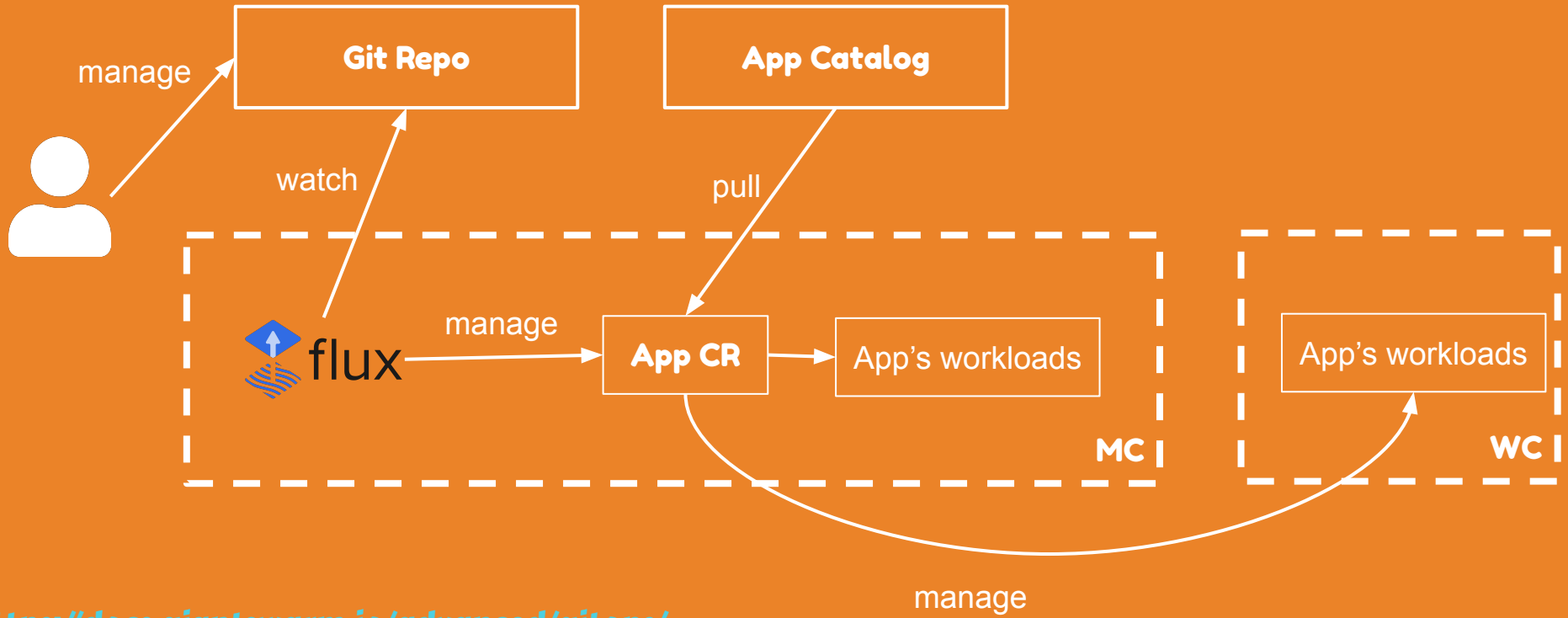
- Observability
  - Prometheus : You can access any WC by just changing a URL param
  - Grafana: There are many built-in dashboard
  - Loki, EFK stack
  - Note: GiantSwarm monitors all clusters and gives 7/24 support.
- Security
  - Kyverno: There are some built-in policies for secure clusters
  - Falco
  - Trivy
- Connectivity
  - linkerd
  - Kong, ingress

Gitops + Flux - I



https://docs.giantswarm.io/advanced/gitops/

Gitops + Flux - 2



https://docs.giantswarm.io/advanced/gitops/

Gitops + Flux - 3: Fleets & Collections

e.g. https://github.com/giantswarm/openstack-app-collection

Management Tools

- Powerful internal admin tools
  - Lists management clusters
  - Get kubeconfig for MC/WC
  - Get access to MC/WC
  - Daily ops activities (deploy, rollout, drain etc.)
  - Open MC apps like prometheus, grafana etc.
- We have powerful user tools like kubectl-gs
  - Easy templating
  - WC management, especially GS types (e.g. apps)
- Web UI (giantswarm/happa)

User Experience

Open source tools in upstream repositories are good but not enough for a smooth user experience. We

- implement additional/helper operators
  - for easy upgrades with declarative approach like GitOps
  - for backup
  - for cleanup
- create useful helm charts (See `cluster-$provider` repos in giantswarm org)

The problem domain is huge and complex.

Ops is inevitable.

Support is must.

for listening!