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.

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.

A diagram of a cluster

Description automatically generated

**Questions?**

* How can I consistently provision machines, load balancers, VPC, etc., across multiple infrastructure providers and locations?
* How can I automate cluster lifecycle management, including things like upgrades and cluster deletion?
* How can I scale these processes to manage any number of clusters?

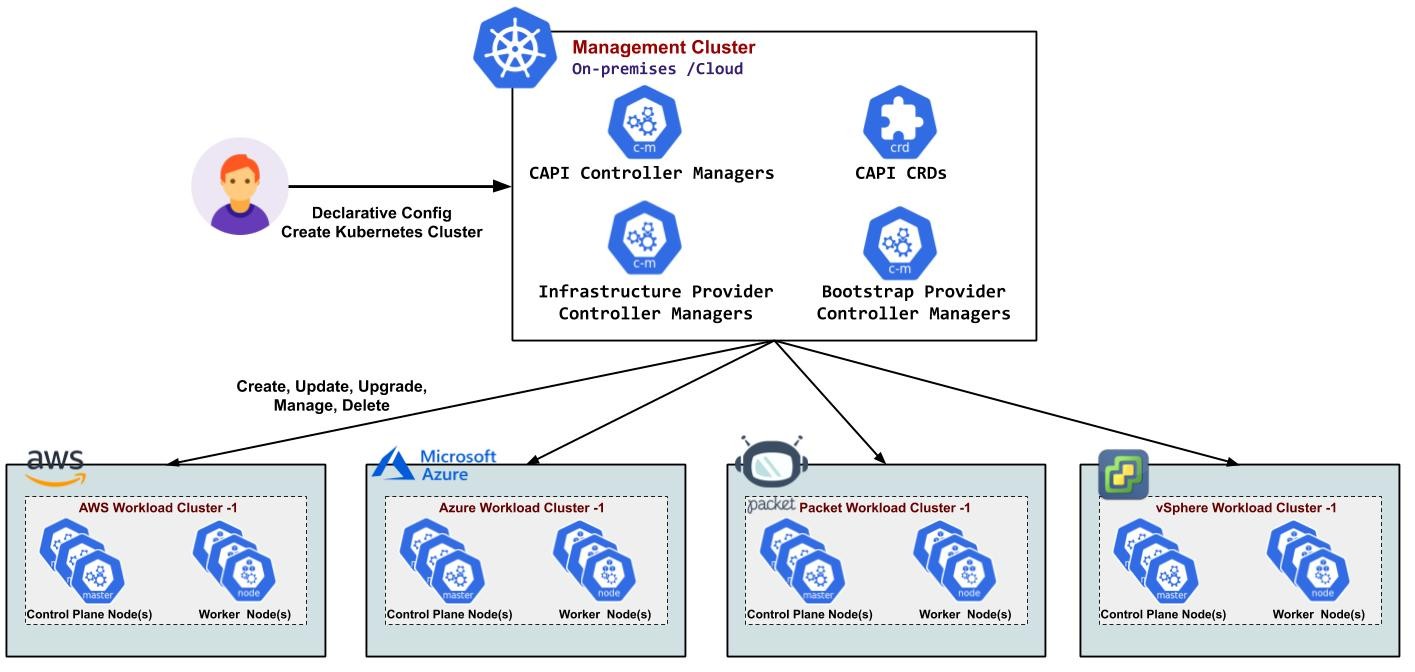
While the project has “**API**” in its name, it is not just an API. It can be looked at as a **framework**. It certainly provides an API, but also comes with a controller that reconciles the cluster — creates, updates, and deletes resources in the cloud. Besides the controller, we have a CLI called clusterctl which is used to create a new cluster from nothing.

Cluster API uses CRDs underneath for provisioning and managing Kubernetes cluster. It comprises of five custom resources on top of Kubernetes cluster.

A blue hexagon with icons

Description automatically generated

Cluster API requires an existing Kubernetes cluster accessible via kubectl. During the installation process the Kubernetes cluster will be transformed into a **management cluster** by installing the **Cluster API provider components**, so it is recommended to keep it separated from any application workload.



**Walkthrough:**

**kind** can be used for creating a local Kubernetes cluster for development environments or for the creation of a temporary bootstrap cluster used to provision a target management cluster on the selected infrastructure provider.

**Step1 - Install Go**

1. In Browser Open: <https://go.dev/dl/>
2. Download Linux Platform file: <https://go.dev/dl/go1.23.1.linux-amd64.tar.gz>
3. cd ~/Downloads
4. sudo rm -rf /usr/local/go
5. sudo tar -C /usr/local -xzf go1.23.1.linux-amd64.tar.gz
6. export PATH=$PATH:/usr/local/go/bin
7. ~~Edit Profile: code ~/.profile~~ 🡪 ~~Add line: PATH=$PATH:/usr/local/go/bin~~ 🡪 ~~Save and Close~~ 🡪 ~~Logout and login~~
8. go version

**Step2 - Install kind:**

go install sigs.k8s.io/kind@v0.24.0

export PATH=$PATH:$(go env GOPATH)/bin

**Step3 - Create a directory for temp files:**

mkdir kinddemo

cd kinddemo

**Step4 - Create docker.yaml**

Run the following command to create a kind config file for allowing the Docker provider to access Docker on the host:

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

networking:

ipFamily: dual

nodes:

- role: control-plane

extraMounts:

- hostPath: /var/run/docker.sock

containerPath: /var/run/docker.sock

**Step5 - Create Cluster:**

kubectl config get-contexts

kind create cluster --config docker.yaml

kubectl cluster-info

**Step6 - Install ClusterCtl:**

curl -L https://github.com/kubernetes-sigs/cluster-api/releases/download/v1.8.2/clusterctl-linux-amd64 -o clusterctl

sudo install -o root -g root -m 0755 clusterctl /usr/local/bin/clusterctl

clusterctl version

**Step7 - Create a Service Principal in Azure**

1. Azure Portal 🡪 search Microsoft Entra ID 🡪 App registrations 🡪 +New registration
2. Name=ClusterAPISP 🡪 Register
   1. Note: Application (client) ID, Directory (tenant)
   2. Go to Certificates & Secrets 🡪 + New client secret 🡪 Add
   3. Note the Value of Secret
3. **Assigned Role=Contributor to Service Principal Identity** 
   1. Go to Subscription 🡪 Access Control (IAM) 🡪 Add 🡪 Add role assignment
   2. Role Tab: Select Privileged administrator roles 🡪 Select **Contributor** 🡪 Next
   3. Members Tab: +Select Members 🡪 search ClusterAPISP 🡪 Select
   4. Review + assign 🡪 Review + assign
4. Use the above details in environmental variables below

**Step8 - Initialization for common providers**

export AZURE\_SUBSCRIPTION\_ID="<SubscriptionId>"

export AZURE\_TENANT\_ID="<Tenant>"

export AZURE\_CLIENT\_ID="<AppId>"

export AZURE\_CLIENT\_ID\_USER\_ASSIGNED\_IDENTITY=$AZURE\_CLIENT\_ID # for compatibility with CAPZ v1.16 templates

export AZURE\_CLIENT\_SECRET="<Password>"

**# Settings needed for AzureClusterIdentity used by the AzureCluster**

export AZURE\_CLUSTER\_IDENTITY\_SECRET\_NAME="cluster-identity-secret"

export CLUSTER\_IDENTITY\_NAME="cluster-identity"

export AZURE\_CLUSTER\_IDENTITY\_SECRET\_NAMESPACE="default"

**Step9 - Initialize the management cluster**

Now that we’ve got clusterctl installed and all the prerequisites in place, let’s transform the Kubernetes cluster into a management cluster by using clusterctl init.

The command accepts as input a list of providers to install; when executed for the first time, clusterctl init automatically adds to the list the cluster-api core provider, and if unspecified, it also adds the kubeadm bootstrap and kubeadm control-plane providers.

**Create a secret to include the password of the Service Principal identity created in Azure**

**# This secret will be referenced by the AzureClusterIdentity used by the AzureCluster**

kubectl create secret generic "${AZURE\_CLUSTER\_IDENTITY\_SECRET\_NAME}" --from-literal=clientSecret="${AZURE\_CLIENT\_SECRET}" --namespace "${AZURE\_CLUSTER\_IDENTITY\_SECRET\_NAMESPACE}"

**# Finally, initialize the management cluster**

clusterctl init --infrastructure azure

**Enabling Feature Gates**

export CLUSTER\_TOPOLOGY=true

**Step10 - Create Managed Identity in Azure**

1. Azure Portal 🡪 Seach Managed Identity 🡪 + Create

Create New Resource Group: Name=**capz-ci**

Name = **cloud-provider-user-identity**

A screenshot of a computer

Description automatically generated

**Step11 - Assigned Role=Contributor to Managed Identity**

1. Go to Subscription 🡪 Access Control (IAM) 🡪 Add 🡪 Add role assignment
2. Role Tab: Select Privileged administrator roles 🡪 Select **Contributor** 🡪 Next
3. Members Tab: +Select Members 🡪 search **cloud-provider-user-identity** 🡪 Select
4. Review + assign 🡪 Review + assign

Create your first workload cluster

**Make sure you choose a VM size which is available in the desired location for your subscription.**

az vm list-skus -l eastus -r virtualMachines -o table

**Step12 - Preparing the workload cluster configuration**

# Name of the Azure datacenter location. Change this value to your desired location.

export AZURE\_LOCATION="eastus"

# Select VM types.

export AZURE\_CONTROL\_PLANE\_MACHINE\_TYPE="Standard\_D2s\_v3"

export AZURE\_NODE\_MACHINE\_TYPE="Standard\_D2s\_v3"

# [Optional] Select resource group. The default value is ${CLUSTER\_NAME}.

export AZURE\_RESOURCE\_GROUP="ClusterAPIDemo-rg"

**Step13 - Generating the cluster configuration**

For the purpose of this tutorial, we’ll name our cluster capi-quickstart.

**clusterctl generate** cluster **capi-demo** --infrastructure azure --kubernetes-version v1.29.7 --control-plane-machine-count=1 --worker-machine-count=2 > capi-quickstart.yaml

Note: This creates a YAML file named capi-quickstart.yaml with a predefined list of Cluster API objects; Cluster, Machines, Machine Deployments, etc.

**Step13 - Cluster templates authenticate with Workload Identity by default. Modify the AzureClusterIdentity for ServicePrincipal authentication.**

sudo snap install yq

yq -i "with(. | select(.kind == \"AzureClusterIdentity\"); .spec.type |= \"ServicePrincipal\" | .spec.clientSecret.name |= \"${AZURE\_CLUSTER\_IDENTITY\_SECRET\_NAME}\" | .spec.clientSecret.namespace |= \"${AZURE\_CLUSTER\_IDENTITY\_SECRET\_NAMESPACE}\")" capi-quickstart.yaml

**Step14 - Apply the workload cluster**

kubectl apply -f capi-quickstart.yaml

**Note** **that below** **Resources are created by above command:**

1. Cluster
2. KubeadmControlPlane
3. AzureMachineTemplate
4. MachineDeployment
5. AzureMachineTemplate
6. KubeadmConfigTemplate
7. AzureClusterIdentity

**Step15 - Accessing the workload cluster**

kubectl get cluster

clusterctl describe cluster capi-demo

kubectl get kubeadmcontrolplane

kubectl logs deploy/capz-controller-manager -n capz-system manager

**Step16 – After the first control plane node is up and running, we can retrieve the workload cluster Kubeconfig.**

clusterctl get kubeconfig capi-quickstart > capi-demo.kubeconfig

**Step17 - Install the official cloud-provider-azure Helm chart on the workload cluster:**

helm install --kubeconfig=./capi-quickstart.kubeconfig --repo https://raw.githubusercontent.com/kubernetes-sigs/cloud-provider-azure/master/helm/repo cloud-provider-azure --generate-name --set infra.clusterName=capi-demo --set cloudControllerManager.clusterCIDR="192.168.0.0/16"

**Step18 - Deploy a CNI Solution**

The control plane **won’t be Ready** until we install a CNI in the next step.

helm repo add projectcalico https://docs.tigera.io/calico/charts --kubeconfig=./capi-quickstart.kubeconfig && \

helm install calico projectcalico/tigera-operator --kubeconfig=./capi-quickstart.kubeconfig -f https://raw.githubusercontent.com/kubernetes-sigs/cluster-api-provider-azure/main/templates/addons/calico/values.yaml --namespace tigera-operator --create-namespace

**Step19 -** After a short while, our nodes should be running and in Ready state

kubectl --kubeconfig=./capi-quickstart.kubeconfig get nodes