

---

## Virtual Kubelet



Virtual Kubelet is an open source Kubernetes kubelet implementation that masquerades as a kubelet for the purposes of connecting Kubernetes to other APIs. This allows the nodes to be backed by other services like ACI, AWS Fargate, IoT Edge, Tensile Kube etc. The primary scenario for VK is enabling the extension of the Kubernetes API into serverless container platforms like ACI and Fargate, though we are open to others. However, it should be noted that VK is explicitly not intended to be an alternative to Kubernetes federation.

Virtual Kubelet features a pluggable architecture and direct use of Kubernetes primitives, making it much easier to build on.

We invite the Kubernetes ecosystem to join us in empowering developers to build upon our base. Join our slack channel named, virtual-kubelet, within the Kubernetes slack group.

The best description is “Kubernetes API on top, programmable back.”

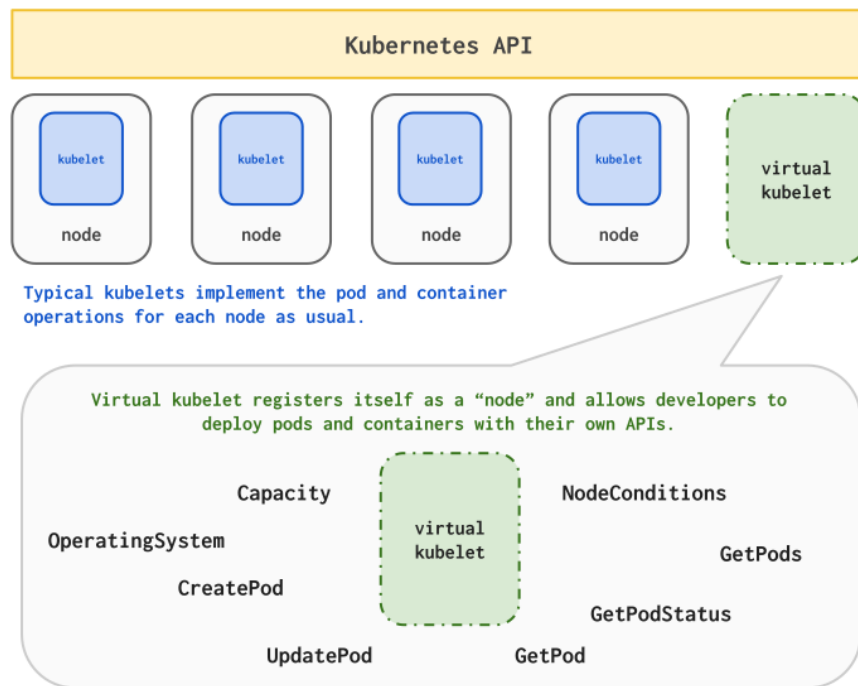
### Table of Contents

- How It Works
- Usage
- Providers
  - Admiralty Multi-Cluster Scheduler
  - Alibaba Cloud ECI Provider
  - Azure Container Instances Provider
  - Azure Batch GPU Provider
  - AWS Fargate Provider
  - Elotl Kip
  - HashiCorp Nomad
  - Ligo
  - OpenStack Zun
  - Tensile Kube Provider
  - Adding a New Provider via the Provider Interface
- Testing
  - Unit tests
  - End-to-end tests
- Known quirks and workarounds

- 
- Contributing

## How It Works

The diagram below illustrates how Virtual-Kubelet works.



## Usage

Virtual Kubelet is focused on providing a library that you can consume in your project to build a custom Kubernetes node agent.

See godoc for up to date instructions on consuming this project: <https://godoc.org/github.com/virtual-kubelet/virtual-kubelet>

There are implementations available for several providers, see those repos for details on how to deploy.

## Current Features

- create, delete and update pods
- container logs, exec, and metrics

- 
- get pod, pods and pod status
  - capacity
  - node addresses, node capacity, node daemon endpoints
  - operating system
  - bring your own virtual network

## Providers

This project features a pluggable provider interface developers can implement that defines the actions of a typical kubelet.

This enables on-demand and nearly instantaneous container compute, orchestrated by Kubernetes, without having VM infrastructure to manage and while still leveraging the portable Kubernetes API.

Each provider may have its own configuration file, and required environmental variables.

Providers must provide the following functionality to be considered a supported integration with Virtual Kubelet. 1. Provides the back-end plumbing necessary to support the lifecycle management of pods, containers and supporting resources in the context of Kubernetes. 2. Conforms to the current API provided by Virtual Kubelet. 3. Does not have access to the Kubernetes API Server and has a well-defined callback mechanism for getting data like secrets or configmaps.

## Admiralty Multi-Cluster Scheduler

Admiralty Multi-Cluster Scheduler mutates annotated pods into “proxy pods” scheduled on a virtual-kubelet node and creates corresponding “delegate pods” in remote clusters (actually running the containers). A feedback loop updates the statuses and annotations of the proxy pods to reflect the statuses and annotations of the delegate pods. You can find more details in the Admiralty Multi-Cluster Scheduler documentation.

## Alibaba Cloud ECI Provider

Alibaba Cloud ECI(Elastic Container Instance) is a service that allow you run containers without having to manage servers or clusters.

You can find more details in the Alibaba Cloud ECI provider documentation.

**Configuration File** The alibaba ECI provider will read configuration file specified by the `--provider-config` flag.

The example configure file is in the ECI provider repository.

---

## Azure Container Instances Provider

The Azure Container Instances Provider allows you to utilize both typical pods on VMs and Azure Container instances simultaneously in the same Kubernetes cluster.

You can find detailed instructions on how to set it up and how to test it in the Azure Container Instances Provider documentation.

**Configuration File** The Azure connector can use a configuration file specified by the `--provider-config` flag. The config file is in TOML format, and an example lives in `providers/azure/example.toml`.

## AWS Fargate Provider

AWS Fargate is a technology that allows you to run containers without having to manage servers or clusters.

The AWS Fargate provider allows you to deploy pods to AWS Fargate. Your pods on AWS Fargate have access to VPC networking with dedicated ENIs in your subnets, public IP addresses to connect to the internet, private IP addresses to connect to your Kubernetes cluster, security groups, IAM roles, CloudWatch Logs and many other AWS services. Pods on Fargate can co-exist with pods on regular worker nodes in the same Kubernetes cluster.

Easy instructions and a sample configuration file is available in the AWS Fargate provider documentation. Please note that this provider is not currently supported.

## Elotl Kip

Kip is a provider that runs pods in cloud instances, allowing a Kubernetes cluster to transparently scale workloads into a cloud. When a pod is scheduled onto the virtual node, Kip starts a right-sized cloud instance for the pod's workload and dispatches the pod onto the instance. When the pod is finished running, the cloud instance is terminated.

When workloads run on Kip, your cluster size naturally scales with the cluster workload, pods are strongly isolated from each other and the user is freed from managing worker nodes and strategically packing pods onto nodes.

---

## HashiCorp Nomad Provider

HashiCorp Nomad provider for Virtual Kubelet connects your Kubernetes cluster with Nomad cluster by exposing the Nomad cluster as a node in Kubernetes. By using the provider, pods that are scheduled on the virtual Nomad node registered on Kubernetes will run as jobs on Nomad clients as they would on a Kubernetes node.

For detailed instructions, follow the guide [here](#).

## Liqo Provider

Liqo implements a provider for Virtual Kubelet designed to transparently offload pods and services to “peered” Kubernetes remote cluster. Liqo is capable of discovering neighbor clusters (using DNS, mDNS) and “peer” with them, or in other words, establish a relationship to share part of the cluster resources. When a cluster has established a peering, a new instance of the Liqo Virtual Kubelet is spawned to seamlessly extend the capacity of the cluster, by providing an abstraction of the resources of the remote cluster. The provider combined with the Liqo network fabric extends the cluster networking by enabling Pod-to-Pod traffic and multi-cluster east-west services, supporting endpoints on both clusters.

For detailed instruction, follow the guide [here](#)

## OpenStack Zun Provider

OpenStack Zun provider for Virtual Kubelet connects your Kubernetes cluster with OpenStack in order to run Kubernetes pods on OpenStack Cloud. Your pods on OpenStack have access to OpenStack tenant networks because they have Neutron ports in your subnets. Each pod will have private IP addresses to connect to other OpenStack resources (i.e. VMs) within your tenant, optionally have floating IP addresses to connect to the internet, and bind-mount Cinder volumes into a path inside a pod’s container.

```
1 ./bin/virtual-kubelet --provider="openstack"
```

For detailed instructions, follow the guide [here](#).

## Tensile Kube Provider

Tensile kube is contributed by tencent games, which is provider for Virtual Kubelet connects your Kubernetes cluster with other Kubernetes clusters. This provider enables us extending Kubernetes to

---

an unlimited one. By using the provider, pods that are scheduled on the virtual node registered on Kubernetes will run as jobs on other Kubernetes clusters' nodes.

### Adding a New Provider via the Provider Interface

Providers consume this project as a library which implements the core logic of a Kubernetes node agent (Kubelet), and wire up their implementation for performing the necessary actions.

There are 3 main interfaces:

**PodLifecycleHandler** When pods are created, updated, or deleted from Kubernetes, these methods are called to handle those actions.

godoc#PodLifecycleHandler

```
1 type PodLifecycleHandler interface {
2     // CreatePod takes a Kubernetes Pod and deploys it within the
3     // provider.
4     CreatePod(ctx context.Context, pod *corev1.Pod) error
5     // UpdatePod takes a Kubernetes Pod and updates it within the
6     // provider.
7     UpdatePod(ctx context.Context, pod *corev1.Pod) error
8     // DeletePod takes a Kubernetes Pod and deletes it from the
9     // provider.
10    DeletePod(ctx context.Context, pod *corev1.Pod) error
11    // GetPod retrieves a pod by name from the provider (can be cached)
12    // .
13    GetPod(ctx context.Context, namespace, name string) (*corev1.Pod,
14        error)
15    // GetPodStatus retrieves the status of a pod by name from the
16    // provider.
17    GetPodStatus(ctx context.Context, namespace, name string) (*corev1.
18        PodStatus, error)
19    // GetPods retrieves a list of all pods running on the provider (
20    // can be cached).
21    GetPods(context.Context) ([]*corev1.Pod, error)
22 }
```

There is also an optional interface **PodNotifier** which enables the provider to asynchronously notify the virtual-kubelet about pod status changes. If this interface is not implemented, virtual-kubelet will periodically check the status of all pods.

---

It is highly recommended to implement `PodNotifier`, especially if you plan to run a large number of pods.

godoc#PodNotifier

```
1 type PodNotifier interface {
2     // NotifyPods instructs the notifier to call the passed in function
      when
3     // the pod status changes.
4     //
5     // NotifyPods should not block callers.
6     NotifyPods(context.Context, func(*corev1.Pod))
7 }
```

`PodLifecycleHandler` is consumed by the `PodController` which is the core logic for managing pods assigned to the node.

```
1 pc, _ := node.NewPodController(podControllerConfig) // <--
      instatiates the pod controller
2 pc.Run(ctx) // <-- starts watching for pods to be scheduled on the
      node
```

**NodeProvider** NodeProvider is responsible for notifying the virtual-kubelet about node status updates. Virtual-Kubelet will periodically check the status of the node and update Kubernetes accordingly.

godoc#NodeProvider

```
1 type NodeProvider interface {
2     // Ping checks if the node is still active.
3     // This is intended to be lightweight as it will be called
      periodically as a
4     // heartbeat to keep the node marked as ready in Kubernetes.
5     Ping(context.Context) error
6
7     // NotifyNodeStatus is used to asynchronously monitor the node.
8     // The passed in callback should be called any time there is a
      change to the
9     // node's status.
10    // This will generally trigger a call to the Kubernetes API server
      to update
11    // the status.
12    //
13    // NotifyNodeStatus should not block callers.
14    NotifyNodeStatus(ctx context.Context, cb func(*corev1.Node))
15 }
```

Virtual Kubelet provides a `NaiveNodeProvider` that you can use if you do not plan to have custom

---

node behavior.

godoc#NaiveNodeProvider

`NodeProvider` gets consumed by the `NodeController`, which is core logic for managing the node object in Kubernetes.

```
1    nc, _ := node.NewNodeController(nodeProvider, nodeSpec) // <--  
    instantiate a node controller from a node provider and a  
    kubernetes node spec  
2    nc.Run(ctx) // <-- creates the node in kubernetes and starts up he  
    controller
```

**API endpoints** One of the roles of a Kubelet is to accept requests from the API server for things like `kubectl logs` and `kubectl exec`. Helpers for setting this up are provided here

**Scrape Pod metrics** If you want to use HPA(Horizontal Pod Autoscaler) in your cluster, the provider should implement the `GetStatsSummary` function. Then metrics-server will be able to get the metrics of the pods on virtual-kubelet. Otherwise, you may see `No metrics for pod` on metrics-server, which means the metrics of the pods on virtual-kubelet are not collected.

## Testing

### Unit tests

Running the unit tests locally is as simple as `make test`.

### End-to-end tests

Check out `test/e2e` for more details.

## Known quirks and workarounds

### Missing Load Balancer IP addresses for services

**Providers that do not support service discovery** Kubernetes 1.9 introduces a new flag, `ServiceNodeExclusion`, for the control plane's Controller Manager. Enabling this flag in the Controller Manager's manifest allows Kubernetes to exclude Virtual Kubelet nodes from being added to Load Balancer pools, allowing you to create public facing services with external IPs without issue.



---

**Workaround** Cluster requirements: Kubernetes 1.9 or above

Enable the ServiceNodeExclusion flag, by modifying the Controller Manager manifest and adding `--feature-gates=ServiceNodeExclusion=true` to the command line arguments.

## Contributing

Virtual Kubelet follows the CNCF Code of Conduct. Sign the CNCF CLA to be able to make Pull Requests to this repo.

Monthly Virtual Kubelet Office Hours are held at 10am PST on the second Thursday of every month in this zoom meeting room. Check out the calendar [here](#).

Our google drive with design specifications and meeting notes are [here](#).

We also have a community slack channel named virtual-kubelet in the Kubernetes slack. You can also connect with the Virtual Kubelet community via the mailing list.