Virtual Kubelet

Overview

StackPath's Virtual Kubelet provider allows you to leverage the power of Kubernetes (K8s) to seamlessly deploy and manage your applications across StackPath's expansive Edge Compute network from the control plane of your choice, increasing scalability and reliability, while decreasing latency.

This feature enables you to use the Kubernetes control plane to create and manage pods as you normally would, without having to worry about managing your own hardware and infrastructure, as StackPath's Virtual Kubelet provider takes care of scheduling these pods for you on our Edge Compute nodes.

This guide will explain how to create and configure StackPath Edge Compute containers using Virtual Kubelet.

Key Features

- **Volumes using *csi***. Mount volumes in your pods using the *csi* volume type with the driver *virtual-kubelet.storage.compute.edgeengine.io*.
- **Environment variables**. Set environment variables for your pods using the Kubernetes *env* field in your pod specification.
- **Instance size selection**. Specify resource requirements for your pods using the Kubernetes *resources* field in your pod specification.
- **Startup, liveness and readiness probes**.
  Configure *startup*, *liveness* and *readiness* probes for your pods using the Kubernetes *livenessProbe* and *readinessProbe* fields in your pod specification.
- **Private images using image pull secrets**. Use Kubernetes image pull secrets to securely pull private container images from a registry using the Kubernetes *imagePullSecrets* field in your pod specification.

Getting Started

The following are required before you can start using the StackPath Edge Compute Virtual Kubelet provider:

- A Kubernetes cluster. This is where you will be configuring Virtual Kubelet.
- A StackPath account
- API Credentials

Creating a Virtual Kubelet Pod

The instructions below explain how to deploy a Kubernetes deployment for StackPath's Virtual Kubelet Provider using Kustomize.

Usage

1. Confirm that Kustomize is installed in your environment by running the *kustomize version* command. If you haven't already installed Kustomize, follow the instructions here.

   To find the Kustomize version embedded in recent versions of kubectl, run *kubectl version*:

   kubectl version --short --client
   Client Version: v1.26.0
   Kustomize Version: v4.5.7

2. Clone this repository to your local environment.
3. Navigate to the *base* directory, which contains the base Virtual Kubelet deployment:

   bash cd deployment/kustomize/base

4. Follow this guide to obtain StackPath API credentials and update the *config.properties* file with your StackPath account, Stack, client, and secret IDs:

   SP_STACK_ID = {your-stack-id}
   SP_CLIENT_ID = {your-client-id}
   SP_CLIENT_SECRET = {your-client-secret}

5. To deploy the Virtual Kubelet resources, run the following command:

   kubectl apply -k

   This will create the Virtual Kubelet deployment in your Kubernetes cluster. Please note that a secret will be generated from the *config.properties* file specified in the *secretGenerator* section of the *kustomization.yaml* file. This secret contains the values of the environment variables specified in the *config.properties* file.

Updating Resources

To customize the Virtual Kubelet deployment, create an overlay directory (*vk-deployment-updated* in this example) within the *overlays* directory with a *kustomization.yaml* file that specifies the changes you want to make:

```
.
├── base
│   ├── cluster-role.yaml
│   ├── config.properties
│   ├── kustomization.yaml
│   ├── namespace.yaml
│   ├── service-account.yaml
│   └── vk-deployment.yaml
└── overlays
    └── vk-deployment-updated
        └── kustomization.yaml
```

Create the following *kustomization.yaml* file under the *overlay* directory to create a Virtual Kubelet in a namespace other than the default one while updating the values of *SP_CITY_CODE* and *SP_STACK_ID* environment variables. We will be using *sp-atl* as the location for this example:

```
resources:
- ../../base

namespace: sp-atl

images:
- name: stackpath.com/virtual-kubelet
newTag: 0.0.2

configMapGenerator:
- name: sp-vk-location
behavior: replace
literals:
- SP_CITY_CODE=ATL

secretGenerator:
- name: sp-vk-secrets
behavior: merge
literals:
- SP_STACK_ID= <another_stack_id>
```

- *resources* references the base resources that are inherited by this overlay, which includes a default Virtual Kubelet deployment configuration.
- *namespace* specifies that the Virtual Kubelet deployment will be created in the *sp-atl* namespace.
- *images* is used to define the version of the StackPath Virtual Kubelet image to be used.

- *configMapGenerator* replaces the existing value of *SP_CITY_CODE* with *ATL*, which specifies the geographic location of the Edge Compute infrastructure.
- *secretGenerator* merges the existing *config.properties* file with a new *SP_STACK_ID* value of *<another_stack_id>*. This updates the StackPath Stack ID specified in *config.properties*.

To deploy overlay, run the following command:

```
kubectl apply -k overlays/vk-deployment-updated
```

Creating a Workload

Now that you've created a Virtual Kubelet pod using the steps above, you're ready to move on to the next step. Once this pod is running, you can then create a standard pod and StackPath workload.

To use the Virtual Kubelet deployment to deploy workloads in the StackPath Edge Compute infrastructure, configure your pods to use the *virtual-kubelet.io/provider* toleration and *type: virtual-kubelet* node selector.

Here is an example configuration that will create the simplest possible container in the *sp-atl* namespace by providing only a name (*my-pod*) and image (*my-image*):

```
apiVersion: v1
kind: Pod
metadata:
 name: my-pod
 namespace: sp-atl
spec:
 containers:
 - name: my-container
   image: my-image
tolerations:
 - key: virtual-kubelet.io/provider
   operator: Equal
   value: stackpath
   effect: NoSchedule
nodeSelector:
   kubernetes.io/role: agent
   type: virtual-kubelet
```

You can customize your workload by adding more configurations under the *specs* field, as if you were using the StackPath API.

Here is what a more standard workload configuration's YAML file would look like:

```
apiVersion: v1
kind: Pod
metadata:
  name: webserver
  namespace: vk-sp
spec:
  containers:
    - name: webserver
      image: nginx:latest
      args:
        - "example1"
        - "example2"
      command:
        - "nginx"
      ports:
        - name: http
          containerPort: 80
        - name: https
          containerPort: 443
      env:
        - name: VAR
          value: val
      resources:
        requests:
          memory: "1Gi"
          cpu: "250m"
        limits:
          memory: "4Gi"
          cpu: "2"

      volumeMounts:
        - mountPath: "/disk-1"
          name: volume-1
      livenessProbe:
        tcpSocket:
          port: 80
        initialDelaySeconds: 5
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /
          port: 80
```

```
        httpHeaders:
          - name: Custom-Header
            value: Custom
      initialDelaySeconds: 5
      periodSeconds: 10
      successThreshold: 2
      timeoutSeconds: 10
      failureThreshold: 1

  volumes:
    - name: volume-1
      csi:
        driver: virtual-kubelet.storage.compute.edgeengine.io
        volumeAttributes:
          size: "2Gi"

  tolerations:
    - key: virtual-kubelet.io/provider
      operator: Equal
      value: stackpath
      effect: NoSchedule
  nodeSelector:
    kubernetes.io/role: agent
    type: virtual-kubelet
```

Using the example above, let's create a workload. The name of our YAML file is *my_example_pod.yaml*. It's located in our *sp/testing* folder. Using kubectl, run the following command:

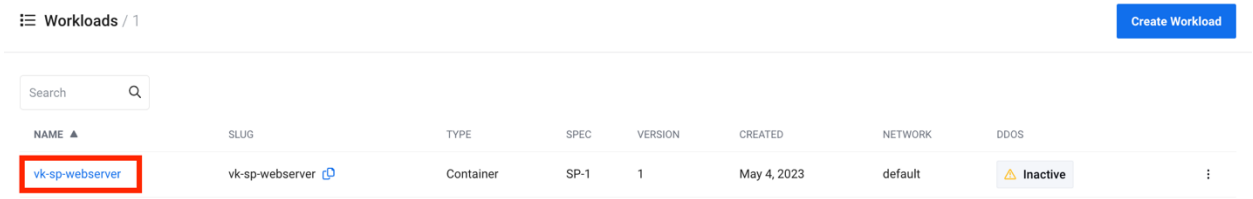
```
kubectl apply -f sp/testing/my_example_pod.yaml
```

This command creates a Virtual Kubelet container workload using the configuration defined in our YAML file.

Validating our Workload

We can confirm that this workload has been created and is running properly by checking either of the following:

- The Edge Compute Dashboard in StackPath Control Portal:



- Get all workloads via the StackPath API. Retrieve the appropriate workload ID and use it to get our workload's detailed information.

Enabling Remote Management for Pods

To enable remote management for pods, you can use the workload.platform.stackpath.net/remote-management annotation in the pod definition metadata. By setting this annotation to true, the remote management capabilities for the containers listed in the pod will be enabled.

To enable remote management, add the following annotation to your pod definition metadata:

annotations:

workload.platform.stackpath.net/remote-management: "true"

By default, if this annotation is not provided or set to "false", remote management will be disabled.

For more information on Edge Compute Workload Metadata and other terms related to StackPath Edge Compute, please refer to Learn Edge Compute Terms.

Enabling remote management should be done with caution and only for trusted pods or in controlled environments where appropriate security measures are in place.

Limitations

StackPath Edge Compute currently supports eight instance types: SP-1 through SP-8. Our Kubernetes provider will launch the smallest instance that provides the resources defined in the pod specification YAML file (within the *resources* parameter). If the pod specification requires more resources than what is available in the SP-8 instance, the provider will provision the SP-8 instance type.

Here are the specifications for each of the available instance types:

Subscription	Cores	RAM
SP-1	1	2GiB
SP-2	2	4GiB
SP-3	2	8GiB
SP-4	4	16GiB
SP-5	8	32GiB
SP-6	16	64GiB
SP-7	32	128GiB
SP-8	48	256GiB

Supported PodSpec File Fields

The following is a comprehensive list of supported fields in the PodSpec file when using StackPath's Virtual Kubelet Provider for Edge Compute:

- **shareProcessNamespace**: Allows multiple containers in a pod to share the same process namespace.
- **hostAliases**: Specifies custom host-to-IP mappings for the pod.
- **dnsConfig**: Configures DNS settings for the pod.
- **securityContext**: Defines security-related settings for the containers in the pod, including permissions and access levels.
    - **runAsUser**: Specifies the user ID that runs the container.
    - **runAsGroup**: Specifies the primary group ID of the container.
    - **runAsNonRoot**: Ensures that the container does not run as root.
    - **supplementalGroups**: Lists additional group IDs applied to the container.
    - **sysctls**: Configures kernel parameters for the container.
- **containers**: Specifies the main containers in the pod.
    - **name**: Specifies the name of the container.
    - **image**: Specifies the container image.
    - **command**: Specifies the command to be run in the container.
    - **args**: Specifies the arguments to be passed to the container command.
    - **ports**: Configures ports for the container.
    - **env**: Sets environment variables for the container.
    - **resources**: Specifies the resource requirements and limits for the container.
    - **securityContext** (Container-specific):
        - **runAsUser**: Specifies the user ID that runs the container.
        - **runAsGroup**: Specifies the primary group ID of the container.

        - **runAsNonRoot**: Ensures that the container does not run as root.
        - **allowPrivilegeEscalation**: Allows privilege escalation for the container.
        - **capabilities**: Specifies Linux capabilities for the container.
    - **volumeMounts**: Mounts volumes into the container.
    - **startupProbe**: Configures the startup probe for the container.
    - **livenessProbe**: Configures the liveness probe for the container.
    - **readinessProbe**: Configures the readiness probe for the container.
    - **lifecycle**:
        - **postStart**: Executed after the container starts.
        - **preStop**: Executed before the container is terminated due to any reason.
    - **imagePullPolicy**: Specifies when to pull the container image (we currently support Always and IfNotPresented)
    - **workingDir**: Sets the working directory inside the container.
    - **terminationMessagePath**: Specifies the path to the container termination message.
    - **terminationMessagePolicy**: Specifies how the termination message should be populated.
- **initContainers**: Defines one or more containers that should run before the main containers in the pod (supports same fields as **containers**)
- **volumes**: Configures volumes to be used in the pod.