Custom Kubernetes Node Types for Karpenter Autoscaling

Document Level Classification

300

Intended audience: Development Teams and DevOps Engineers

Kubernetes cluster autoscaling is handled by Karpenter. Deployments to our clusters are managed by our ArgoCD repository, but only Substrate engineers have access to approve and push out changes (they could effect every cluster in the company if done wrong). If you would like a new pool of nodes attached to your cluster of a specific node type or architecture, this guide will walk you through that process.

Making Changes to Karpenter Nodepools

Before making changes to how autoscaling works in your cluster, you need to know about the Karpenter CRD used to configure it. This custom resource used to define autoscaling of nodes by Karpenter is called 'nodepool' and the documentation for it is here: https://karpenter.sh/docs/concepts/nodepools/

The overall configuration process goes in the following order, but you will only be making changes to the Wrapper Helm Chart with the intent that the Karpenter Nodepool CRDs in your cluster will be changed. If you want to see the current configuration(s) in your cluster, you can run `kubectl get nodepool -A -o yaml`.

```
ArgoCD Repository  --> Wrapper Helm Chart (by Epic) --> Karpenter Helm Chart --> Karpenter NodePool CRD
```

Custom Nodepool Creation

To configure a new nodepool, you will need to branch the ArgoCD repository. Once your branch has been created, find your cluster in the cluster-configs directory. This file controls the cluster-specific overrides values sent to all Helm charts that are sent to your cluster. In this file we can provision an additional nodepool for use with your cluster.

See the following example of a custom nodepool. You are able to nest any value in this Karpenter wrapper chart under the `karpenter:` section below. You can see the values supported by the chart in the template here. In this example, the new nodepool will be named custom (`kubectl get nodepool custom`) and will use `amd64` architecture nodes in the instanceType values list.

If you wanted to use GPU accelerated nodes, you would simply change the node type values to AWS GPU node types. If you wanted ARM nodes, you would change the architecture to `arm64` . To see a lit of all things that can be filtered on, check the supported labels here.

To see a list of the resources you can request by instanceType, go here and here.

Dedicated ARM64 Example

dead-dev-dontwork.yaml

```
karpenter:
  values:
    nodepools:
      custom:
        enabled: true
        taints:
          - key: epicgames.com/my-special-taint
            effect: NoExecute
        labels:
          use: dedicated
          app: jenkins
        requirements:
          zone:
            values: []
          arch:
            values:
              - amd64
          capacityType:
            values:
              - on-demand
          instanceType:
            values:
              - c5.9xlarge
              - c5.12xlarge
              - m5a.12xlarge
              - c6i.8xlarge
              - c6i.12xlarge
```

```
            - c6a.8xlarge
            - c6a.12xlarge
            - r6i.8xlarge
            - r6i.12xlarge
            - r5.8xlarge
            - r5.12xlarge
            - r5a.12xlarge
        nodeClassRef:
          name: default
        disruption:
          consolidationPolicy: WhenEmptyOrUnderutilized
```

NVIDIA GPU Example

It is feasible to *not* include a `taints` section as is done in the example below. Given that Karpenter will provision nodes based on the lowest node pricing at the time of pods being Pending, this should work out fine for GPU nodes. However, if you have a mix of system architectures without taints or explicit node selectors on all your workloads, you will find that pod will schedule onto nodes of the wrong architecture and thereafter will fail to start. In short, you can leave taints off of nodepools that use GPU acceleration, but you should not remove the taints for nodepools that are of non-amd64 architecture or else your pods will unexpectedly schedule onto nodes of the wrong architecture.

WARNING

It appears that currently instance types starting with p3. or p4. do not properly contain the "nvidia.com/gpu" resource. If you want to use NVIDIA GPUs, please use the instance types beginning with g5.

dead-dev-dontwork.yaml

```
# Remember to check that you have enough instance quota for GPU nodes!
karpenter:
```

```
  values:
    nodepools:
```

```
      enabled: true
      labels:
        use: not-mining
      taints:
        - key : epicgames.com/gpu
          effect: NoExecute
      requirements:
        zone:
          values: []
        arch:
          values:
            - amd64
        capacityType:
          values:
            - on-demand
        instanceType:
          values: []
        instanceCategory:
          values:
            - g
        instanceGPUManufacturer:
          values:
            - nvidia
      nodeClassRef:
        name: default
      disruption:
        consolidationPolicy: WhenEmptyOrUnderutilized
```

You can see an example of a deployment that consumes GPUs here.

Assigning Workloads to Your Tainted Nodes

When a pod with a node selector is detected as Pending, Karpenter will automatically determine which nodes need to be provisioned to satisfy the demands of that pod. This means that your next step is to configure the workloads that you want to provision onto your new tainted Karpenter nodes. The Assigning Workloads to Tainted Nodes article has all the information you need to do that.

Deploying Your Nodepool

Once your nodepool and workload specs are written, make a pull request to your cluster spec in the substrate/argocd repository. Hosting team may be able to help you out. A member with access to the ArgoCD UI will verify your pull request, merge it, and deploy your change to your cluster.

As soon as your new nodepool is in place, which you can check with `kubectl get nodepool`, you can safely merge your updated spec with tolerations and node label selectors. Now, when new pods are Pending, Karpenter will automatically provision tainted nodes for your workload that no other workloads will be able to use!
