Workloads

Your workloads are deployed in containers, which are deployed in Pods in Kubernetes. A Pod includes one or more containers. Typically, one or more Pods that provide the same service are deployed in a Kubernetes service. Once you've deployed multiple Pods that provide the same service, you can:

- View information about the workloads (p. 490) running on each of your clusters using the AWS Management Console.
- Vertically scale Pods up or down with the Kubernetes Vertical Pod Autoscaler (p. 326).
- Horizontally scale the number of Pods needed to meet demand up or down with the Kubernetes Horizontal Pod Autoscaler (p. 330).
- Create an external (for internet-accessible Pods) or an internal (for private Pods) network load balancer (p. 332) to balance network traffic across Pods. The load balancer routes traffic at Layer 4 of the OSI model.
- Create an Application load balancing on Amazon EKS (p. 338) to balance application traffic across Pods. The application load balancer routes traffic at Layer 7 of the OSI model.
- If you're new to Kubernetes, this topic helps you Deploy a sample application (p. 319).
- You can restrict IP addresses that can be assigned to a service (p. 342) with `externalIPs`.

Deploy a sample application

In this topic, you deploy a sample application to your cluster.

Prerequisites

- An existing Kubernetes cluster with at least one node. If you don't have an existing Amazon EKS cluster, you can deploy one using one of the Getting started with Amazon EKS (p. 4) guides. If you're deploying a Windows application, then you must have Windows support (p. 46) enabled for your cluster and at least one Amazon EC2 Windows node.
- `Kubectl` installed on your computer. For more information, see Installing or updating `kubectl` (p. 4).
- `Kubectl` configured to communicate with your cluster. For more information, see Creating or updating a `kubeconfig` file for an Amazon EKS cluster (p. 393).
- If you plan to deploy your sample workload to Fargate, then you must have an existing Fargate profile (p. 130) that includes the same namespace created in this tutorial, which is `eks-sample-app`, unless you change the name. If you used one of the getting started guides (p. 4) to create your cluster, then you'll have to create a new profile, or add the namespace to your existing profile, because the profile created in the getting started guides doesn't specify the namespace used in this tutorial. Your VPC must also have at least one private subnet.

To deploy a sample application

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes Pods, deployments, and services, you can experiment with changing other values.

1. Create a namespace. A namespace allows you to group resources in Kubernetes. For more information, see Namespaces in the Kubernetes documentation. If you plan to deploy your sample application to AWS Fargate (p. 124), make sure that the value for `namespace` in your AWS Fargate profile (p. 130) is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

2. Create a Kubernetes deployment. This sample deployment pulls a container image from a public repository and deploys three replicas (individual Pods) of it to your cluster. To learn more, see Deployments in the Kubernetes documentation. You can deploy the application to Linux or Windows nodes. If you're deploying to Fargate, then you can only deploy a Linux application.

   a. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see Storage (p. 208).

      Linux

      The `amd64` or `arm64` `values` under the `kubernetes.io/arch` key mean that the application can be deployed to either hardware architecture (if you have both in your cluster). This is possible because this image is a multi-architecture image, but not all are. You can determine the hardware architecture that the image is supported on by viewing the image details in the repository that you're pulling it from. When deploying images that don't support a hardware architecture type, or that you don't want the image deployed to, remove that type from the manifest. For more information, see Well-Known Labels, Annotations and Taints in the Kubernetes documentation.

      The `kubernetes.io/os: linux` `nodeSelector` means that if you had Linux and Windows nodes (for example) in your cluster, the image would only be deployed to Linux nodes. For more information, see Well-Known Labels, Annotations and Taints in the Kubernetes documentation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eks-sample-linux-deployment
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-linux-app
  template:
    metadata:
      labels:
        app: eks-sample-linux-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
      nodeSelector:
        kubernetes.io/os: linux
```

Windows

The `kubernetes.io/os: windows` `nodeSelector` means that if you had Windows and Linux nodes (for example) in your cluster, the image would only be deployed to Windows nodes. For more information, see Well-Known Labels, Annotations and Taints in the Kubernetes documentation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eks-sample-windows-deployment
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-windows-app
  template:
    metadata:
      labels:
        app: eks-sample-windows-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: beta.kubernetes.io/arch
                operator: In
                values:
                - amd64
      containers:
      - name: windows-server-iis
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        command:
        - powershell.exe
        - -command
        - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\
\wwwroot\\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
      nodeSelector:
        kubernetes.io/os: windows
```

b. Apply the deployment manifest to your cluster.

```
kubectl apply -f eks-sample-deployment.yaml
```

3. Create a service. A service allows you to access all replicas through a single IP address or name. For more information, see Service in the Kubernetes documentation. Though not implemented in the sample application, if you have applications that need to interact with other AWS services, we recommend that you create Kubernetes service accounts for your Pods, and associate them to AWS

IAM accounts. By specifying service accounts, your Pods have only the minimum permissions that you specify for them to interact with other services. For more information, see IAM roles for service accounts (p. 425).

a. Save the following contents to a file named `eks-sample-service.yaml`. Kubernetes assigns the service its own IP address that is accessible only from within the cluster. To access the service from outside of your cluster, deploy the AWS Load Balancer Controller (p. 292) to load balance application (p. 338) or network (p. 332) traffic to the service.

   Linux

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: eks-sample-linux-service
     namespace: eks-sample-app
     labels:
       app: eks-sample-linux-app
   spec:
     selector:
       app: eks-sample-linux-app
     ports:
       - protocol: TCP
         port: 80
         targetPort: 80
   ```

   Windows

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: eks-sample-windows-service
     namespace: eks-sample-app
     labels:
       app: eks-sample-windows-app
   spec:
     selector:
       app: eks-sample-windows-app
     ports:
       - protocol: TCP
         port: 80
         targetPort: 80
   ```

b. Apply the service manifest to your cluster.

   ```
   kubectl apply -f eks-sample-service.yaml
   ```

4. View all resources that exist in the `eks-sample-app` namespace.

   ```
   kubectl get all -n eks-sample-app
   ```

   The example output is as follows.

   If you deployed Windows resources, then all instances of *`linux`* in the following output are `windows`. The other *`example values`* may be different from your output.

   ```
   NAME                                               READY   STATUS    RESTARTS   AGE
   pod/eks-sample-linux-deployment-65b7669776-m6qxz   1/1     Running   0          27m
   pod/eks-sample-linux-deployment-65b7669776-mmxvd   1/1     Running   0          27m
   pod/eks-sample-linux-deployment-65b7669776-qzn22   1/1     Running   0          27m
   ```

```
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
 AGE
service/eks-sample-linux-service   ClusterIP   10.100.74.8     <none>        80/TCP
 32m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-linux-deployment 3/3     3            3           27m

NAME                                                    DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-linux-deployment-776d8f8fd8    3         3         3
 27m
```

In the output, you see the service and deployment that were specified in the sample manifests deployed in previous steps. You also see three Pods. This is because `3 replicas` were specified in the sample manifest. For more information about Pods, see Pods in the Kubernetes documentation. Kubernetes automatically creates the `replicaset` resource, even though it isn't specified in the sample manifests. For more information about `ReplicaSets`, see ReplicaSet in the Kubernetes documentation.

Note
Kubernetes maintains the number of replicas that are specified in the manifest. If this were a production deployment and you wanted Kubernetes to horizontally scale the number of replicas or vertically scale the compute resources for the Pods, use the Horizontal Pod Autoscaler (p. 330) and the Vertical Pod Autoscaler (p. 326) to do so.

5. View the details of the deployed service. If you deployed a Windows service, replace *`linux`* with **`windows`**.

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

The example output is as follows.

If you deployed Windows resources, then all instances of *`linux`* in the following output are `windows`. The other *`example values`* may be different from your output.

```
Name:              eks-sample-linux-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-linux-app
Annotations:       <none>
Selector:          app=eks-sample-linux-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

In the previous output, the value for `IP:` is a unique IP address that can be reached from any node or Pod within the cluster, but it can't be reached from outside of the cluster. The values for `Endpoints` are IP addresses assigned from within your VPC to the Pods that are part of the service.

6. View the details of one of the Pods listed in the output when you viewed the namespace (p. 322) in a previous step. If you deployed a Windows app, replace *`linux`* with **`windows`** and replace *`776d8f8fd8-78w66`* with the value returned for one of your Pods.

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

Abbreviated output

If you deployed Windows resources, then all instances of `linux` in the following output are `windows`. The other *example values* may be different from your output.

```
Name:         eks-sample-linux-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
 ip-192-168-45-132.us-west-2.compute.internal
[...]
```

In the previous output, the value for `IP:` is a unique IP that's assigned to the Pod from the CIDR block assigned to the subnet that the node is in. If you prefer to assign Pods IP addresses from different CIDR blocks, you can change the default behavior. For more information, see Tutorial: Custom networking (p. 265). You can also see that the Kubernetes scheduler scheduled the Pod on the Node with the IP address *192.168.45.132*.

Tip
Rather than using the command line, you can view many details about Pods, services, deployments, and other Kubernetes resources in the AWS Management Console. For more information, see View Kubernetes resources (p. 490).

7. Run a shell on the Pod that you described in the previous step, replacing *65b7669776-m6qxz* with the ID of one of your Pods.

   Linux

   ```
   kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app
    -- /bin/bash
   ```

   Windows

   ```
   kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app
    -- powershell.exe
   ```

8. From the Pod shell, view the output from the web server that was installed with your deployment in a previous step. You only need to specify the service name. It is resolved to the service's IP address by CoreDNS, which is deployed with an Amazon EKS cluster, by default.

   Linux

   ```
   curl eks-sample-linux-service
   ```

   The example output is as follows.

   ```
   <!DOCTYPE html>
   <html>
   <head>
   <title>Welcome to nginx!</title>
   [...]
   ```

   Windows

   ```
   Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
   ```

   The example output is as follows.

   ```
   StatusCode        : 200
   StatusDescription : OK
   Content           : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > <
    H 1 > H e l l o
                         E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l
    >
   ```

9. From the Pod shell, view the DNS server for the Pod.

   Linux

   ```
   cat /etc/resolv.conf
   ```

   The example output is as follows.

   ```
   nameserver 10.100.0.10
   search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
   west-2.compute.internal
   options ndots:5
   ```

   In the previous output, `10.100.0.10` is automatically assigned as the `nameserver` for all Pods deployed to the cluster.

   Windows

   ```
   Get-NetIPConfiguration
   ```

   Abbreviated output

   ```
   InterfaceAlias       : vEthernet
   [...]
   IPv4Address          : 192.168.63.14
   [...]
   DNSServer            : 10.100.0.10
   ```

In the previous output, `10.100.0.10` is automatically assigned as the DNS server for all Pods deployed to the cluster.

10. Disconnect from the Pod by typing `exit`.
11. Once you're finished with the sample application, you can remove the sample namespace, service, and deployment with the following command.

```
kubectl delete namespace eks-sample-app
```

Vertical Pod Autoscaler

The Kubernetes Vertical Pod Autoscaler automatically adjusts the CPU and memory reservations for your Pods to help "right size" your applications. This adjustment can improve cluster resource utilization and free up CPU and memory for other Pods. This topic helps you to deploy the Vertical Pod Autoscaler to your cluster and verify that it is working.

Prerequisites

- You have an existing Amazon EKS cluster. If you don't, see Getting started with Amazon EKS (p. 4).
- You have the Kubernetes Metrics Server installed. For more information, see Installing the Kubernetes Metrics Server (p. 412).
- You are using a `kubectl` client that is configured to communicate with your Amazon EKS cluster (p. 16).
- OpenSSL `1.1.1` or later installed on your device.

Deploy the Vertical Pod Autoscaler

In this section, you deploy the Vertical Pod Autoscaler to your cluster.

To deploy the Vertical Pod Autoscaler

1. Open a terminal window and navigate to a directory where you would like to download the Vertical Pod Autoscaler source code.
2. Clone the kubernetes/autoscaler GitHub repository.

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. Change to the `vertical-pod-autoscaler` directory.

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (Optional) If you have already deployed another version of the Vertical Pod Autoscaler, remove it with the following command.

```
./hack/vpa-down.sh
```

5. If your nodes don't have internet access to the `registry.k8s.io` container registry, then you need to pull the following images and push them to your own private repository. For more information about how to pull the images and push them to your own private repository, see Copy a container image from one repository to another repository (p. 344).

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
```