AWS EKS Deployment Documentation

Table of Contents

Table of Contents

Prerequisites
- Dockerization
- Kubernetes Tools
  - Install kubectl
  - Install AWS IAM Authenticator
  - Install AWS Cli
  - Configure AWS CLI
  - Install eksctl

Create a Kubernetes Cluster with EKS
- Create the cluster with worker nodes
  - Create the configuration file
  - Testing cluster

Pods
- Description
- Creating Pod on cluster

Deployments
- Description
- Creating Deployment

Service
- Description
- Creating service

Reference

Prerequisites

Dockerization

It is basic and mandatory requirement that to get kubernetes for any app to be production ready we have to docarize the app/software first which are going to reside in cluster.

Please create Dockerfile and dockerize the app for production. For more refer https://docs.docker.com/engine/reference/builder/

Kubernetes Tools

Kubernetes clusters require ***kubectl*** and kubelet binaries and AWS EKS cluster requires the ***aws-iam-authenticator*** binary to allow IAM authentication for your Kubernetes cluster.

Create the default ***~/.kube*** directory for storing kubectl configuration, in windows the location will be ***C:\Users\{username}***

```
mkdir -p ~/.kube
```

Install kubectl

```
sudo curl --silent --location -o /usr/local/bin/kubectl
https://amazon-eks.s3-us-west-2.amazonaws.com/1.12.7/2019-03-27/bi
n/linux/amd64/kubectl

sudo chmod +x /usr/local/bin/kubectl
```

For windows machine please refer https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-windows

Install AWS IAM Authenticator

For linux

1. Download the Amazon EKS-vended aws-iam-authenticator binary from Amazon S3:

   ```
   curl -o aws-iam-authenticator
   https://amazon-eks.s3-us-west-2.amazonaws.com/1.12.7/2019-03-27/bi
   n/linux/amd64/aws-iam-authenticator
   ```

2. Apply execute permissions to the binary.

   ```
   chmod +x ./aws-iam-authenticator
   ```

3. Copy the binary to a folder in your $PATH. We recommend creating a *$HOME/bin/aws-iam-authenticator* and ensuring that *$HOME/bin* comes first in your *$PATH*.

   ```
   mkdir -p $HOME/bin && cp ./aws-iam-authenticator
   $HOME/bin/aws-iam-authenticator && export PATH=$HOME/bin:$PATH
   ```

4. Add $HOME/bin to your PATH environment variable.

   ```
   echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
   ```

5. Test that the aws-iam-authenticator binary works.

   ```
   aws-iam-authenticator help
   ```

For Windows

1. Open a PowerShell terminal window and download the Amazon EKS-vended aws-iam-authenticator binary from Amazon S3

   ```
   curl -o aws-iam-authenticator.exe
   https://amazon-eks.s3-us-west-2.amazonaws.com/1.12.7/2019-03-27/bi
   n/windows/amd64/aws-iam-authenticator.exe
   ```

2. Copy the binary to a folder in your PATH. If you have an existing directory in your PATH that you use for command line utilities, copy the binary to that directory. Otherwise, complete the following steps.

    a. Create a new directory for your command line binaries, such as C:\bin.
    b. Copy the aws-iam-authenticator.exe binary to your new directory.
    c. Edit your user or system PATH environment variable to add the new directory to your PATH.
    d. Close your PowerShell terminal and open a new one to pick up the new PATH variable.

3. Test that the aws-iam-authenticator binary works.

```
aws-iam-authenticator help
```

Install AWS Cli

For Windows there is a direct link to get installer

https://docs.aws.amazon.com/cli/latest/userguide/install-windows.html#install-msi-on-windows

For Linux refer

https://docs.aws.amazon.com/cli/latest/userguide/install-linux.html

Configure AWS CLI

After aws cli installation configure AWS CLI with your credentials using

```
aws configure
```

Install eksctl

eksctl is a tool jointly developed by AWS and Weaveworks that automates much of the experience of creating EKS clusters.

For linux

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/download/latest_rel
ease/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

sudo mv -v /tmp/eksctl /usr/local/bin
```

For Windows

Windows users can use chocolatey

```
chocolatey install eksctl
```

Create a Kubernetes Cluster with EKS

Every Kubernates cluster has one master node and other worker nodes, using eks we create master node first and then will configure worker node, both tasks can be done in a single command.

Create the cluster with worker nodes

Create a Kubernetes cluster using the following command.

```
eksctl create cluster --name prod --nodegroup-name
standard-workers --node-type t2.micro --nodes 3 --nodes-min 1
--nodes-max 4 --node-ami auto
```

Note:

*This command will also create **kubeconfig** in **~/.kube** which is very important to kubectl to communicate with cluster in aws eks. In case this file is not created then please look for errors occurred in command prompt while creating cluster. If your cluster is successfully created yet you are missing this configuration file then you have to create this file manually*

Create the configuration file

Once the cluster has moved to the ACTIVE state, download and run the ***create-kubeconfig.sh*** script.

```
aws s3 cp s3://aws-kubernetes-artifacts/v0.5/create-kubeconfig.sh . && \
chmod +x create-kubeconfig.sh && ./create-kubeconfig.sh
```

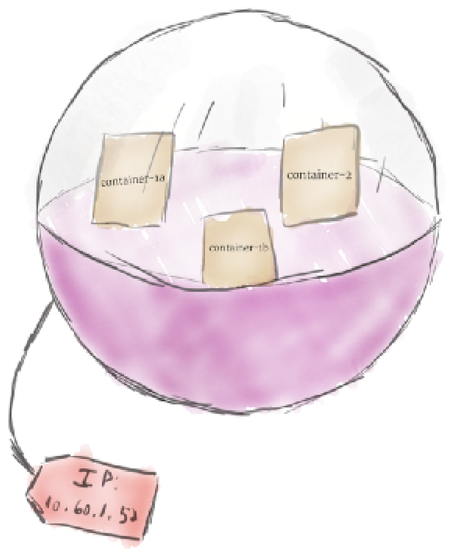
Testing cluster

Confirm your Nodes:

```
kubectl get nodes
```

Pods

(Reference: https://medium.com/google-cloud/kubernetes-101-pods-nodes-containers-and-clusters-c1509e409e16)



Description

Unlike other systems you may have used in the past, Kubernetes doesn't run containers directly; instead it wraps one or more containers into a higher-level structure called a pod. Any containers in the same pod will share the same resources and local network. Containers can easily communicate with other containers in the same pod as though they were on the same machine while maintaining a degree of isolation from others.

Pods are used as the unit of replication in Kubernetes. If your application becomes too popular and a single pod instance can't carry the load, Kubernetes can be configured to deploy new replicas of your pod to the cluster as necessary. Even when not under heavy load, it is standard to have multiple copies of a pod running at any time in a production system to allow load balancing and failure resistance.

Pods can hold multiple containers, but you should limit yourself when possible. Because pods are scaled up and down as a unit, all containers in a pod must scale together, regardless of their individual needs. This leads to wasted resources and an expensive bill. To resolve this, pods should remain as small as possible, typically holding only a main process and its tightly-coupled helper containers (these helper containers are typically referred to as "side-cars").

Creating Pod on cluster

Pods can be directly created/deployed using pre exist container images like below

```
kubectl run nginx --image=nginx
```

To create your own user defined Pod with multiple container or configuration you can use the following

Each resource in Kubernetes can be defined using a configuration file. For example, an NGINX pod can be defined with configuration file shown in below:

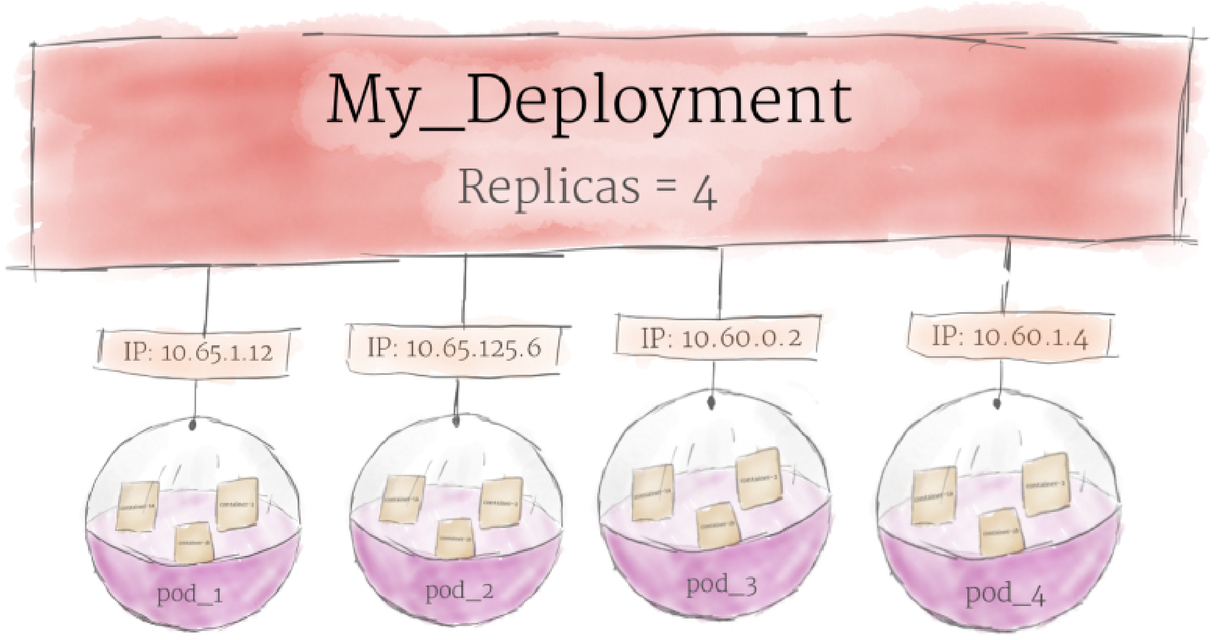
```
Pod_name.Yaml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

To create pod using above configuration use following command

```
kubectl apply -f pod_name.yaml
```

Deployments

(Reference: https://medium.com/google-cloud/kubernetes-101-pods-nodes-containers-and-clusters-c1509e409e16)



Description

Although pods are the basic unit of computation in Kubernetes, they are not typically directly launched on a cluster. Instead, pods are usually managed by one more layer of abstraction: the deployment.

A deployment's primary purpose is to declare how many replicas of a pod should be running at a time. When a deployment is added to the cluster, it will automatically spin up the requested number of pods, and then monitor them. If a pod dies, the deployment will automatically re-create it.

Using a deployment, you don't have to deal with pods manually. You can just declare the desired state of the system, and it will be managed for you automatically.

Creating Deployment

Deployment can be created using configuration file as shown below as an example of nodejs app

```
Deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
 name: kubernates-node
 labels:
   app: kubernates-node
 namespace: default
spec:
 replicas: 1
 selector:
   matchLabels:
     app: kubernates-node
 strategy:
   rollingUpdate:
     maxSurge: 25%
     maxUnavailable: 25%
   type: RollingUpdate
 template:
   metadata:
     labels:
       app: kubernates-node
   spec:
     containers:
     - image: sidgujrathi/kubernates-node:latest
       imagePullPolicy: Always
       name: kubernates-node
       ports:
       - containerPort: 3000
         protocol: TCP
```

To create deployment using this configuration use following command

```
kubectl create -f deployment.yaml
```

Service

Description

A pod is ephemeral. Each pod is assigned a unique IP address. If a pod that belongs to a replication controller dies, then it is recreated and may be given a different IP address. Further, additional pods may be created using Deployment or Replica Set. This makes it difficult for an application server, such as access a database, such as MySQL, using its IP address.

A Service is an abstraction that defines a logical set of pods and a policy by which to access them. The IP address assigned to a service does not change over time, and thus can be relied upon by other pods. Typically, the pods belonging to a service are defined by a label selector. This is similar mechanism to how pods belong to a replica set.

This abstraction of selecting pods using labels enables a loose coupling. The number of pods in the deployment may scale up or down but the application server can continue to access the database using the service.

Creating service

To create service again we will use configuration file

```
Service.yaml

apiVersion: v1
kind: Service
metadata:
  name: kubernates-node
spec:
  selector:
    app: kubernates-node
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8080
  type: LoadBalancer
```

Above service defines service which will create Load Balancer which will be applied to deployment we have created before named *kubernetes-node*
To deploy this service to cluster use following command

```
kubectl create -f service.yaml --record
```

This will create and attach load balancer to deployment and will return one public url to access this deployment using browser publically

Reference

1. https://eksworkshop.com
2. aws-workshop-for-kubernetes/01-path-basics at master · aws-samples/aws-workshop-for-kubernetes
3. Getting Started with eksctl - Amazon EKS
4. https://kubernetes.io/