Kubernetes User's Guide for Altibase

Table of Contents

Introduction

Kubernetes is an open-source container orchestration system that automates software deployment, scaling, and management. A Pod is the smallest execution unit in Kubernetes. This guide provides information on how to create an Altibase Pod using Altibase container image registered at Docker Hub and use it in the Kubernetes v1.20.4 environment. Please refer to the Altibase Docker Guide for information on how to create Altibase docker image and the Kubernetes website for other Kubernetes features.

Creating and Using Pod

Creating Altibase Pod

This section will cover two methods for creating an Altibase Pod. One uses Deployment workload resource, whereas the other creates it manually.

Creating Pod using Deployment workload resource

The following example demonstrates a YAML file creating a Pod composed of a container which starts Altibase server using Deployment and how to monitor the Pod's status.

1. Write YAML File

```
# File Name : altibase-deploy-Pod.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: altibase-deploy-Pod         # Set Deployment Name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: altibase-deploy-Pod      # The Pod the Deployment will manage. This
has to be identical with spec.template.metadata.labels.
  template:
    # From here is Pod template
    metadata:
      labels:
        app: altibase-deploy-Pod    # Set Pod Name
    spec:
      containers:
      - image: altibase/altibase    # Docket Hub Image to be run on the container
        name: altibase              # Container Name
        ports:
        - containerPort: 20300      # Altibase Server Service Port number to be
exposed
          protocol: TCP
        env:
        - name: MODE                # Altibase Server start mode
          value: daemon
     # Until here is Pod template
```

> Environment variables required to start the Altibase server can be added in template.spec.containers.env field. Please refer to the Docker Hub website for more information about the environment variables.

2. Create Pod

```
$ kubectl create -f altibase-deploy-Pod.yaml
Deployment.apps/altibase-deploy-Pod created
```

3. Check the Pod's Status

```
$ kubectl get Pod -o wide
NAME                  READY   STATUS    RESTARTS   AGE   IP              NODE
NOMINATED NODE   READINESS GATES
altibase-deploy-Pod   1/1     Running   0          24s   172.16.135.38   node3
<none>           <none>
```

4. Connect to Altibase Server

```
$ kubectl exec -it altibase-deploy-Pod -- /bin/bash
altibase@altibase-Pod:~$ . set_altibase.env
altibase@altibase-Pod:~$ is
------------------------------------------------------------------
     Altibase Client Query utility.
     Release Version 7.1.0.5.1
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
------------------------------------------------------------------
ISQL_CONNECTION = TCP, SERVER = localhost, PORT_NO = 20300
iSQL> SELECT * FROM TAB;
```

Creating Pod Manually

The following example demonstrates a YAML file manually creating a Pod composed of a container which starts Altibase server and how to monitor the Pod's status.

1. Write YAML File

```
# File Name : altibase-Pod.yaml

apiVersion: v1
kind: Pod
metadata:
  labels:
    app: altibase
  name: altibase-Pod                    # Set Pod Name
spec:
  containers:
  - image: altibase/altibase            # Docker Hub Image to be run on the
Container
    name: altibase
    ports:
    - containerPort: 20300              # Altibase Service Port Number to be
exposed
      protocol: TCP
    env:
    - name: MODE                        # altibase/altibase Image Start Mode
      value: daemon
```

> Environment variables required to start the Altibase server can be added in spec.containers.env field. Please refer to the Docker Hub website for more information about the environment variables.

2. Create Pod

```
$ kubectl create -f altibase-Pod.yaml
Pod/altibase-Pod created
```

3. Check the Pod Status

```
$ kubectl get Pod -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP              NODE
NOMINATED NODE   READINESS GATES
altibase-Pod   1/1     Running   0          11m   172.16.135.37   node3   <none>
          <none>
```

4. Connect to Altibase Server

```
$ kubectl exec -it altibase-Pod -- /bin/bash

altibase@altibase-Pod:~$ . set_altibase.env
altibase@altibase-Pod:~$ is
--------------------------------------------------------------------
     Altibase Client Query utility.
     Release Version 7.1.0.5.1
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
--------------------------------------------------------------------
ISQL_CONNECTION = TCP, SERVER = localhost, PORT_NO = 20300
iSQL> SELECT * FROM TAB;
```

Using Volume

Due to the ephemeral aspect of a Pod, the data are lost when a Pod crashes. To secure the data regardless of the termination of the Pod, Kubernetes provides Volume which is an abstract version of external disks. Kubernetes supports many types of Volumes. For more information about Volume in detail, please refer to Kubernetes website.

Below is an YAML file setting the path of DB data file and online log file on NFS (Network File System) Volume. NFS server has to be configured in advance.

1. Write YAML file

```
# File Name : altibase-deploy-vol-node1.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: altibase-deploy-vol-node1                          # Set Deployment Name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: altibase-node1                                  # The Pod the Deployment
will manage. This has to be identical with spec.template.metadata.labels.
  template:
    # From here is Pod template
```

```
    metadata:
      labels:
        app: altibase-node1                                # Set Pod Name
    spec:
      containers:
      - name: altibase                                     # Container Name
        image: altibase/altibase                           # Docket Hub Image to be
run on the container
        VolumeMounts:
        # Informs where to mount the Volume specified in spec.Volumes.name on the
container
        - name: altibase-nfs-dbs                           # Volume specified in
spec.Volumes.name(for DB data file)
          mountPath: /home/altibase/altibase_home/dbs      # Directory to be used
in altibase-nfs-dbs Volume
        - name: altibase-nfs-logs                          # Volume specified in
spec.Volumes.name(for online log file)
          mountPath: /home/altibase/altibase_home/logs     # Directory to be used
in altibase-nfs-logs Volume
        ports:
        - containerPort: 20300                             # Altbase Service Port
Number to be exposed
          protocol: TCP
        env:
        - name: MODE
          value: daemon
      Volumes:
      - name: altibase-nfs-dbs                             # The Volume to be
provided to the Pod (for DB data file)
        nfs:                                               # altibase-nfs-dbs
Volume type
          server: 192.168.204.139                          # NFS server's IP
address
          path: /home/altibase-nfs/node1/dbs               # NFS server's directory
      - name: altibase-nfs-logs                            # The Volume to be
provided to the Pod (for online log file)
        nfs:                                               # altibase-nfs-logs
Volume Yype
          server: 192.168.204.139                          # NFS Server's IP
Address
          path: /home/altibase-nfs/node1/logs              # NFS Server's Directory
     # Until here is Pod template
```

2. Create Pod

```
$ kubectl create -f altibase-deploy-vol-node1.yaml
Deployment.apps/altibase-deploy-vol-node1 created
```

3. Check Pod Status

```
$ kubectl get Pod -o wide
NAME                                          READY   STATUS    RESTARTS   AGE
IP               NODE        NOMINATED NODE   READINESS GATES
altibase-deploy-vol-node1-7599dcb85b-75jwp   1/1     Running   0          9s
172.16.107.207   k8s-node3   <none>           <none>
```

Using Service

Although Kubernetes assigns Pods their own IP addresses, this is only internal IP addresses of components within a Kubernetes cluster. Every time the Deployment creates a new Pod, a new dynamic IP address is allocated. To access the Pod which dynamically changes, Kubernetes provides Service resource. When Service is created, a static IP address is assigned and the Pod can use its own DNS name which does not change as long as Service exists.

Below is an example creating one Service, two Altibase Pods and connecting to Altibase server from one node to another using the static IP address.

1. Create YAML file

```
# File Name : altibase_Pod_svc.yaml

# altibase-deploy-Pod1 Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: altibase-deploy-Pod1          # Deployment name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: altibase-deploy-Pod1       # The Pod the Deployment will manage. This
has to be identical with spec.template.metadata.labels.
  template:
    # From here is Pod template
    metadata:
      labels:
        app: altibase-deploy-Pod1      # Pod Name
    spec:
      containers:
      - image: altibase/altibase
        name: altibase                 # Container Name
        ports:
        - containerPort: 20300         # Altibase Service Port Number to be
exposed
          protocol: TCP
        env:
        - name: MODE
          value: daemon
    # Until here is Pod template
---

# altibase-deploy-Pod2 Deployment
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: altibase-deploy-Pod2          # Deployment Name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: altibase-deploy-Pod2       # The Pod the Deployment will manage. This
has to be identical with spec.template.metadata.labels.
  template:
    # From here is Pod template
    metadata:
      labels:
        app: altibase-deploy-Pod2     # Pod Name
    spec:
      containers:
      - image: altibase/altibase
        name: altibase                # Container Name
        ports:
        - containerPort: 20300        # Altibase Service Port Number to be
exposed
          protocol: TCP
        env:
        - name: MODE
          value: daemon
     # Until here is Pod template
---

# Service the static IP address will be allocated to
apiVersion: v1
kind: Service
metadata:
  name: altibase-Service       # Service Name
spec:
  ports:
  - port: 20300                # In most cases port and targetport use the same
value
    targetPort: 20300          # The port number the Pod which transmits the
request from the Service is listening to
  selector:
    app: altibase-deploy-Pod1  # The Pod which transmits the request from the
Service
```

2. Create Pod and Service

```
$ kubectl create -f altibase_Pod_svc.yaml
```

3. Check Pod and Service Status

Static IP address 10.110.31.65 is allocated to altibase-service.

```
$ kubectl get Pod,Service -o wide
NAME                                        READY   STATUS    RESTARTS   AGE
IP               NODE        NOMINATED NODE   READINESS GATES
Pod/altibase-deploy-Pod1-68d566b68c-sm6kw   1/1     Running   0          5m43s
172.16.169.145   k8s-node2   <none>           <none>
Pod/altibase-deploy-Pod2-579498ccdb-zv2mg   1/1     Running   0          5m29s
172.16.107.202   k8s-node3   <none>           <none>


NAME                        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE     SELECTOR
Service/altibase-Service    ClusterIP   10.110.31.65   <none>        20300/TCP
4m49s   app=altibase-deploy-Pod1
Service/kubernetes          ClusterIP   10.96.0.1      <none>        443/TCP
15m     <none>
```

4. Connect to Altibase Server with static IP address

Connecting to Altibase server running on altibase-deploy-Pod1 via the static IP address assigned to Service using iSQL on altibase-deploy-Pod2.

```
$ k exec -it altibase-deploy-Pod2-579498ccdb-zv2mg -- /bin/bash

[altibase@altibase-Pod2 ~] $ . set_altibase.env
[altibase@altibase-Pod2 ~] $ is -s 10.110.31.65 -port 20300 -u sys -p manager   #
Use the static IP address allocated to the Service
--------------------------------------------------------------------
     Altibase Client Query utility.
     Release Version 7.1.0.1.6
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
--------------------------------------------------------------------
ISQL_CONNECTION = TCP, SERVER = 10.98.64.213, PORT_NO = 20300
iSQL>
```

Altibase Replication Using Pod

This section covers how to consist Altibase replication node using Deployment and Service. Since a dynamic IP is assigned every time Pod is created, remote host name is used in syntax creating Altibase replication instead of remote server IP address.

The following is an example of creating Pod which has a static DNS name using Deployment and Service which points this Pod, then connecting to Altibase server to create replication object and checking the replication status.

1. Write YAML file creating Deployment

Create two Pods to consist Altibase replication node.

```
# File Name : altibase-deploy-Pod.yaml

# Create altibase-deploy-vol-node1 Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: altibase-deploy-vol-node1  # Deployment Name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: altibase-node1          # The Pod the Deployment will manage. This has
to be identical with spec.template.metadata.labels.
  template:
    # From here is Pod template
    metadata:
      labels:
        app: altibase-node1        # Pod Name
    spec:
      containers:
      - name: altibase
        image: altibase/altibase
        VolumeMounts:
        - name: altibase-nfs-dbs
          mountPath: /home/altibase/altibase_home/dbs
        - name: altibase-nfs-logs
          mountPath: /home/altibase/altibase_home/logs
        ports:
        - containerPort: 20300      # Altibase Service Port to be exposed
          protocol: TCP
        - containerPort: 20301      # Altibase Replication Port to be exposed
          protocol: TCP
        env:
        - name: MODE
          value: replication
        - name: SLAVE_REP_PORT      # Enable Altibase Replication
          value: "20301"            # Altibase Replication Port Number
      Volumes:
      - name: altibase-nfs-dbs
        nfs:
```

```
          server: 192.168.204.139
          path: /home/altibase-nfs/node1/dbs
      - name: altibase-nfs-logs
        nfs:
          server: 192.168.204.139
          path: /home/altibase-nfs/node1/logs

---

# Create altibase-deploy-vol-node2 Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: altibase-deploy-vol-node2     # Deployment Name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: altibase-node2             # The Pod the Deployment will manage. This
has to be identical with spec.template.metadata.labels.
  template:
   # From here is Pod template
    metadata:
      labels:
        app: altibase-node2           # Pod Name
    spec:
      containers:
      - name: altibase
        image: altibase/altibase
        VolumeMounts:
        - name: altibase-nfs-dbs
          mountPath: /home/altibase/altibase_home/dbs
        - name: altibase-nfs-logs
          mountPath: /home/altibase/altibase_home/logs
        ports:
        - containerPort: 20300      # Altibase Service Port to be exposed
          protocol: TCP
        - containerPort: 20301      # Altibase Replication Port to be exposed
          protocol: TCP
        env:
        - name: MODE
          value: replication
        - name: SLAVE_REP_PORT      # Enable Altibase Replication
          value: "20301"            # Altibase Replication Port Number
      Volumes:
      - name: altibase-nfs-dbs
        nfs:
          server: 192.168.204.139
          path: /home/altibase-nfs/node2/dbs
      - name: altibase-nfs-logs
        nfs:
          server: 192.168.204.139
          path: /home/altibase-nfs/node2/logs
```

2. Write YAML file creating Service

Create two Services corresponding to two Pods.

```
# File Name : altibase-Service.yaml

apiVersion: v1
kind: Service
metadata:
  labels:
    app: altibase
  name: altibase-svc-node1     # Service Name
spec:
  ports:
  - name: Service-port
    port: 20300                # In most cases port and targetport use the same
value
    targetPort: 20300
  - name: replication-port
    port: 20301                # In most cases port and targetport use the same
value
    targetPort: 20301
  selector:
    app: altibase-node1        # The Pod which transmits the request from the
Service

---

apiVersion: v1
kind: Service
metadata:
  labels:
    app: altibase
  name: altibase-svc-node2     # Service Name
spec:
  ports:
  - name: Service-port
    port: 20300                # In most cases port and targetport use the same
value
    targetPort: 20300
  - name: replication-port
    port: 20301                # In most cases port and targetport use the same
value
    targetPort: 20301
  selector:
    app: altibase-node2        # The Pod which transmits the request from the
Service
```

3. Create Pod and Service

```
$ kubectl create -f deploy.yaml
Deployment.apps/altibase-deploy-vol-node1 created
Deployment.apps/altibase-deploy-vol-node2 created

$ kubectl create -f Service.yaml
Service/altibase-svc-node1 created
Service/altibase-svc-node2 created
```

4. Check Pod and Service Status

```
$ kubectl get Pod,Service -o wide
NAME                                          READY   STATUS    RESTARTS   AGE
    IP               NODE        NOMINATED NODE   READINESS GATES
Pod/altibase-deploy-vol-node1-74d75695c4-q7459   1/1     Running   0
3m12s   172.16.169.150   k8s-node2   <none>           <none>
Pod/altibase-deploy-vol-node2-677b65857-x4hxm    1/1     Running   0
3m12s   172.16.107.208   k8s-node3   <none>           <none>

NAME                         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
          AGE     SELECTOR
Service/altibase-svc-node1   ClusterIP   10.98.138.97   <none>
20300/TCP,20301/TCP   73s     app=altibase-node1
Service/altibase-svc-node2   ClusterIP   10.101.72.3    <none>
20300/TCP,20301/TCP   73s     app=altibase-node2
Service/kubernetes           ClusterIP   10.96.0.1      <none>        443/TCP
```

5. Create and Start Replication Object

Create a replication object with the same name as the Service on each Pod's Altibase container and start it.

	altibase-node1 Pod's Altibase container	**altibase-node2 Pod's Altibase container**
Create replication table	CREATE TABLE t1 (c1 INTEGER PRIMARY KEY, c2 INTEGER);	CREATE TABLE t1 (c1 INTEGER PRIMARY KEY, c2 INTEGER);
Create replication object	CREATE REPLICATION rep1 WITH **'altibase-svc-node2'**, 20301 FROM sys.t1 TO sys.t1;	CREATE REPLICATION rep1 WITH **'altibase-svc-node1'**, 20301 FROM sys.t1 TO sys.t1;
Start replication	ALTER REPLICATION rep1 START;	ALTER REPLICATION rep1 START;

6. Check the replication status

Check if the remote server is updated after inserting data into both servers.

	altibase-node1 Pod's Altibase container	**altibase-node2 Pod's Altibase container**
Insert data on altibase-node1	iSQL> INSERT INTO t1 VALUES(1, 1); 1 row inserted.	
Check data in both servers	iSQL> SELECT * FROM t1; C1 C2 ---------- ---------- 1 1 1 row selected.	iSQL> SELECT * FROM t1; C1 C2 ---------- ---------- 1 1 1 row selected.
Insert data on altibase-node2		iSQL> INSERT INTO t1 VALUES (2, 2); 1 row inserted.
Check data in both servers	iSQL> SELECT * FROM t1; C1 C2 ---------- ---------- 1 1 2 2 2 rows selected.	iSQL> SELECT * FROM t1; C1 C2 ---------- ---------- 1 1 2 2 2 rows selected.