此文档包括在 K8S 上部署 和在本地部署 TiDB 集群的流程。

1. 在 Kubernetes 上部署

部署到自托管的 Kubernetes。

集群环境要求

1.1 配置 Storage Class

TiDB 集群中 PD、TiKV、监控等组件以及 TiDB Binlog 和备份等工具都需要使用将数据持久化的存储。

1.1.1 TiDB 集群推荐存储类型

TiKV 自身借助 Raft 实现了数据复制，出现节点故障后，PD 会自动进行数据调度补齐缺失的数据副本，同时 TiKV 要求存储有较低的读写延迟，所以生产环境强烈推荐使用本地 SSD 存储。

PD 同样借助 Raft 实现了数据复制，但作为存储集群元信息的数据库，并不是 IO 密集型应用，所以一般本地普通 SAS 盘或网络 SSD 存储（例如 AWS 上 gp2 类型的 EBS 存储卷，GCP 上的持久化 SSD 盘）就可以满足要求。

监控组件以及 TiDB Binlog、备份等工具，由于自身没有做多副本冗余，所以为保证可用性，推荐用网络存储。其中 TiDB Binlog 的 pump 和 drainer 组件属于 IO 密集型应用，需要较低的读写延迟，所以推荐用高性能的网络存储（例如 AWS 上的 io1 类型的 EBS 存储卷，GCP 上的持久化 SSD 盘）。

在利用 TiDB Operator 部署 TiDB 集群或者备份工具的时候，需要持久化存储的组件都可以通过 values.yaml 配置文件中对应的 `storageClassName` 设置存储类型。不设置时默认都使用 k8s 集群中默认存储类型。

Kubernetes 当前支持静态分配的本地存储。可使用 local-static-provisioner 项目中的 `local-volume-provisioner` 程序创建本地存储对象。`local-volume-provisioner.yaml` 示例文件：

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "local-storage"
provisioner: "kubernetes.io/no-provisioner"
volumeBindingMode: "WaitForFirstConsumer"

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-provisioner-config
  namespace: kube-system
data:
  setPVOwnerRef: "true"
  nodeLabelsForPV: |
```

```
    - kubernetes.io/hostname
  storageClassMap: |
    local-storage:
      hostDir: /workspace/tidb
      mountDir: /workspace/tidb

---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: local-volume-provisioner
  namespace: kube-system
  labels:
    app: local-volume-provisioner
spec:
  selector:
    matchLabels:
      app: local-volume-provisioner
  template:
    metadata:
      labels:
        app: local-volume-provisioner
    spec:
      serviceAccountName: local-storage-admin
      containers:
        - image: "quay.io/external_storage/local-volume-provisioner:v2.3.4"
          name: provisioner
          securityContext:
            privileged: true
          env:
          - name: MY_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: MY_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: JOB_CONTAINER_IMAGE
            value: "quay.io/external_storage/local-volume-
provisioner:v2.3.4"
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
            limits:
              cpu: 100m
              memory: 100Mi
          volumeMounts:
            - mountPath: /etc/provisioner/config
              name: provisioner-config
              readOnly: true
            # mounting /dev in DinD environment would fail
            # - mountPath: /dev
            #   name: provisioner-dev
            - mountPath: /workspace/tidb
              name: local-disks
              mountPropagation: "HostToContainer"
```

```yaml
      volumes:
        - name: provisioner-config
          configMap:
            name: local-provisioner-config
        # - name: provisioner-dev
        #   hostPath:
        #     path: /dev
        - name: local-disks
          hostPath:
            path: /workspace/tidb

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: local-storage-admin
  namespace: kube-system

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-storage-provisioner-pv-binding
  namespace: kube-system
subjects:
- kind: ServiceAccount
  name: local-storage-admin
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: system:persistent-volume-provisioner
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: local-storage-provisioner-node-clusterrole
  namespace: kube-system
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-storage-provisioner-node-binding
  namespace: kube-system
subjects:
- kind: ServiceAccount
  name: local-storage-admin
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: local-storage-provisioner-node-clusterrole
  apiGroup: rbac.authorization.k8s.io

```

部署 `local-volume-provisioner` 程序。

```
# 部署 local-volume-provisioner 程序
kubectl apply -f ./local-volume-provisioner.yaml

# 查看 local-volume-provisioner 程序 pod 运行情况，运行正常即可
kubectl get po -n kube-system -l app=local-volume-provisioner

# 查看PV，可能无输出
kubectl get pv | grep local-storage # 此命令无输出

# 出错可删除
kubectl delete po -n kube-system -l app=local-volume-provisioner
```

1.2 部署 TiDB Operator

1.2.1 准备环境

TiDB Operator 部署前，请确认以下软件需求：

- Kubernetes v1.12 或者更高版本
- DNS 插件
- PersistentVolume
- RBAC 启用（可选）
- Helm 3

1.2.2 创建 CRD

TiDB Operator 使用 Custom Resource Definition (CRD) 扩展 Kubernetes，所以要使用 TiDB Operator，必须先创建 `TidbCluster` 自定义资源类型。只需要在 Kubernetes 集群上创建一次即可：

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-operator/master/manifests/crd.yaml
```

如果服务器没有外网，需要先用有外网的机器下载 `crd.yaml` 文件，然后再进行安装：

```
wget https://raw.githubusercontent.com/pingcap/tidb-operator/master/manifests/crd.yaml
kubectl apply -f ./crd.yaml
```

如果显示如下信息表示 CRD 安装成功：

```
kubectl get crd
NAME                                  CREATED AT
backups.pingcap.com                   2020-06-11T07:59:40Z
backupschedules.pingcap.com           2020-06-11T07:59:41Z
restores.pingcap.com                  2020-06-11T07:59:40Z
tidbclusterautoscalers.pingcap.com    2020-06-11T07:59:42Z
tidbclusters.pingcap.com              2020-06-11T07:59:38Z
tidbinitializers.pingcap.com          2020-06-11T07:59:42Z
tidbmonitors.pingcap.com              2020-06-11T07:59:41Z
```

1.2.3 安装 TiDB Operator

TiDB Operator 使用 Helm 3 安装。

添加 PingCAP 仓库

```
helm repo add pingcap https://charts.pingcap.org/
```

添加完成后，可以使用 `helm search` 搜索 PingCAP 提供的 chart：

```
helm search repo pingcap
```

为 TiDB Operator 创建一个命名空间

```
kubectl create namespace tidb-admin
```

安装 TiDB Operator

```
helm install --namespace tidb-admin tidb-operator pingcap/tidb-operator --
version v1.2.4
```

期望输出：

```
NAME: tidb-operator
LAST DEPLOYED: Mon Jun  1 12:31:43 2020
NAMESPACE: tidb-admin
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Make sure tidb-operator components are running:

    kubectl get pods --namespace tidb-admin -l
app.kubernetes.io/instance=tidb-operator
```

使用以下命令检查 TiDB Operator 组件是否运行起来：

```
kubectl get pods --namespace tidb-admin -l app.kubernetes.io/instance=tidb-
operator
```

期望输出：

```
NAME                                       READY   STATUS    RESTARTS   AGE
tidb-controller-manager-6d8d5c6d64-b8lv4   1/1     Running   0          2m22s
tidb-scheduler-644d59b46f-4f6sb            2/2     Running   0          2m22s
```

当所有的 pods 都处于 Running 状态时，可进行下一步操作。

1.3 配置 TiDB 集群

1.3.1 在 Kubernetes 中配置 TiDB 集群

配置生产可用的 TiDB 集群，涵盖以下内容：

- 资源配置
- 部署配置
- 高可用配置

通过配置 `TidbCluster` CR 来配置 TiDB 集群。参考 TidbCluster 示例和 API 文档（示例和 API 文档请切换到当前使用的 TiDB Operator 版本）完成 TidbCluster CR(Custom Resource)。

1.3.2 集群名称

通过更改 `TiDBCuster` CR 中的 `metadata.name` 来配置集群名称。

1.3.3 版本

正常情况下，集群内的各组件应该使用相同版本，所以一般建议配置 `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.baseImage` + `spec.version` 即可。如果需要为不同的组件配置不同的版本，则可以配置 `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`。

相关参数的格式如下：

- `spec.version`，格式为 `imageTag`，例如 `v5.2.1`
- `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.baseImage`，格式为 `imageName`，例如 `pingcap/tidb`
- `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`，格式为 `imageTag`，例如 `v5.2.1`

1.3.4 推荐配置

configUpdateStrategy

建议设置 `spec.configUpdateStrategy: RollingUpdate`，开启配置自动更新特性，在每次配置更新时，自动对组件执行滚动更新，将修改后的配置应用到集群中。

enableDynamicConfiguration

建议通过设置 `spec.enableDynamicConfiguration: true` 配置 TiKV 的 `--advertise-status-addr` 启动参数。

版本支持：TiDB v4.0.1 及更高版本。

pvReclaimPolicy

建议设置 `spec.pvReclaimPolicy: Retain`，确保 PVC 被删除后 PV 仍然保留，保证数据安全。

mountClusterClientSecret

PD 和 TiKV 支持配置 `mountClusterClientSecret`。如果开启了集群组件间 TLS 支持，建议配置 `spec.pd.mountClusterClientSecret: true` 和 `spec.tikv.mountClusterClientSecret: true`，这样 TiDB Operator 会自动将 `${cluster_name}-cluster-client-secret` 证书挂载到 PD 和 TiKV 容器，方便使用 `pd-ctl` 和 `tikv-ctl`。

测试环境`tidb-cluster.yaml`示例:

主要修改TiDB各组件配置（副本数，存储等内容）。**注意：其中的`config: |`配合`[]`形式的声明使用，猜测可能不能与常规缩进式混用（比如 spec.pd.config. log 的呈现的缩进形式，可能要去掉 config 后的 |），待确定。**

```yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-cluster
  namespace: tidb-cluster

spec:
  version: "v5.2.1"
  timezone: Asia/Shanghai
  configUpdateStrategy: RollingUpdate
  imagePullPolicy: IfNotPresent
  enablePVReclaim: false
  pvReclaimPolicy: Retain

  hostNetwork: false

  ## specify resource requirements for discovery deployment
  discovery:
    limits:
      cpu: "0.2"
    requests:
      cpu: "0.2"

  schedulerName: tidb-scheduler

  enableDynamicConfiguration: true

  ############################
  # TiDB Cluster Components #
  ############################

  pd:
    baseImage: pingcap/pd
    config: |
      [replication]
        enable-placement-rules = true
    ## 注意：config: | 和 [] 形式的声明使用，猜测可能不能与缩进式（比如 log 的缩进形式）混用，待确定
    # config: |
    #   lease = 3
    #   enable-prevote = true
    #   log:
    #     file:
    #       filename: /var/log/pdlog/pd.log
    #     level: "warn"
    #   [replication]
    #     enable-placement-rules = true
    # ## The desired replicas
    replicas: 2
    requests:
      storage: 1Gi
```

```
    mountClusterClientSecret: true

    # if storageClassName is not set, the default Storage Class of the Kubernetes cluster will be used
    storageClassName: local-storage
    ## defines additional volumes for which PVCs will be created by StatefulSet controller
    storageVolumes:
    # this will be suffix of PVC names in VolumeClaimTemplates of PD StatefulSet
    # storageVolume.name：PV 的名称
    - name: pd
      storageClassName: local-storage
      storageSize: 1Gi
      mountPath: /var/log/pdlog


  tidb:
    baseImage: pingcap/tidb
    config: {}
    # config: |
    #   level = "info"
    #   enable-timestamp = true
    #   split-table = true
    #   oom-action = "log"
    #   log:
    #     file:
    #       filename: /var/log/tidblog/tidb.log
    #     level: "warn"
    ## The desired replicas
    replicas: 2
    service:
      type: NodePort
      # Ref: https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/#preserving-the-client-source-ip
      externalTrafficPolicy: Local

    storageClassName: local-storage
    storageVolumes:
    - name: tidb
      storageClassName: local-storage
      storageSize: 1Gi
      mountPath: /var/log/tidblog


  tikv:
    baseImage: pingcap/tikv
    config: {}
    # config: |
    #   prevote = true
    #   storage:
    #     # In basic examples, you can set this to avoid using too much storage.
    #     reserve-space: "0MB"
    #   [storage]
    #     [storage.block-cache]
    #       capacity = "1GB"
    #     rocksdb:
```

```
    #       ##wal-dir: "/var/lib/tikv/wal"
    #     titan:
    #       ##dirname: "/var/lib/titan/data"
    ## The desired replicas
    replicas: 3
    requests:
      storage: 1Gi
    mountClusterClientSecret: true
    separateRocksDBLog: true
    separateRaftLog: true

    storageClassName: local-storage
    storageVolumes:
    - name: tikv
      storageClassName: local-storage
      storageSize: 1Gi
      mountPath: /var/lib/data
    # - name: titan-pvc
    #   storageSize: "1Gi"
    #   mountPath: "/var/lib/titan/data"

  tiflash:
    baseImage: pingcap/tiflash
    version: "v5.2.1"
    replicas: 2
    maxFailoverCount: 3
    storageClaims:
      - resources:
          requests:
            storage: 1Gi
        storageClassName: local-storage
    config: {}
    # config: |
    #   [flash]
    #     [flash.flash_cluster]
    #       log = "/data0/logs/flash_cluster_manager.log"
    #   [logger]
    #     count = 10
    #     level = "information"
    #     errorlog = "/data0/logs/error.log"
    #     log = "/data0/logs/server.log"

```

1.4 部署 TiDB 集群

在标准的 Kubernetes 集群上通过 TiDB Operator 部署 TiDB 集群。

1.4.1 前置条件

- TiDB Operator 部署完成。

1.4.2 部署 TiDB 集群

在部署 TiDB 集群之前，需要先配置 TiDB 集群。请参阅在 Kubernetes 中配置 TiDB 集群。

配置 TiDB 集群后，请按照以下步骤部署 TiDB 集群：

1. 创建 `Namespace`：

```
kubectl create namespace ${namespace}

# 测试环境示例
kubectl create namespace tidb-cluster
```

> **注意：**
>
> `namespace` 是命名空间，可以起一个方便记忆的名字，比如和 `cluster_name` 相同的名称。

部署 TiDB 集群：

```
kubectl apply -f ${cluster_name} -n ${namespace}

# 测试环境示例
kubectl apply -f tidb-cluster/tidb-cluster.yaml -n tidb-cluster
```

> **注意：**
>
> 建议在 `cluster_name` 目录下组织 TiDB 集群的配置，并将其另存为 `${cluster_name}/tidb-cluster.yaml`。默认条件下，修改配置不会自动应用到 TiDB 集群中，只有在 Pod 重启时，才会重新加载新的配置文件。

通过下面命令查看 Pod 状态：

```
kubectl get po -n ${namespace} -l app.kubernetes.io/instance=${cluster_name}

# 测试环境示例
kubectl get po -n tidb-cluster
watch kubectl get po -n tidb-cluster
kubectl get svc tidb-cluster-tidb -n tidb-cluster
kubectl get po -n tidb-cluster -l app.kubernetes.io/instance=tidb-cluster
```

1.5 初始化 TiDB 集群(可选)

对 Kubernetes 上的集群进行初始化配置完成初始化账号和密码设置，以及批量自动执行 SQL 语句对数据库进行初始化。

> **注意：**
>
> - 如果 TiDB 集群创建完以后手动修改过 `root` 用户的密码，初始化会失败。
> - 以下功能只在 TiDB 集群创建后第一次执行起作用，执行完以后再修改不会生效。

1.5.1 配置 TidbInitializer

请参考 TidbInitializer 示例和 API 文档（示例和 API 文档请切换到当前使用的 TiDB Operator 版本）以及下面的步骤，完成 TidbInitializer CR，保存到文件 `${cluster_name}/tidb-initializer.yaml`。

设置集群的命名空间和名称

在 `${cluster_name}/tidb-initializer.yaml` 文件中，修改 `spec.cluster.namespace` 和 `spec.cluster.name` 字段:

```
# ...
spec:
  # ...
  cluster:
    namespace: ${cluster_namespace}
    name: ${cluster_name}
```

初始化账号和密码设置

集群创建时默认会创建 `root` 账号，但是密码为空，这会带来一些安全性问题。可以通过如下步骤为 `root` 账号设置初始密码：

通过下面命令创建 Secret 指定 root 账号密码：

```
kubectl create secret generic tidb-secret --from-literal=root=${root_password} --namespace=${namespace}
```

如果希望能自动创建其它用户，可以在上面命令里面再加上其他用户的 username 和 password，例如：

```
kubectl create secret generic tidb-secret --from-literal=root=${root_password} --from-literal=developer=${developer_password} --namespace=${namespace}
```

该命令会创建 `root` 和 `developer` 两个用户的密码，存到 `tidb-secret` 的 Secret 里面。并且创建的普通用户 `developer` 默认只有 `USAGE` 权限，其他权限请在 `initSql` 中设置。

在 `${cluster_name}/tidb-initializer.yaml` 中设置 `passwordSecret: tidb-secret`。

设置允许访问 TiDB 的主机

在 `${cluster_name}/tidb-initializer.yaml` 中设置 `permitHost: ${mysql_client_host_name}` 配置项来设置允许访问 TiDB 的主机 **host_name**。如果不设置，则允许所有主机访问。详情请参考 MySQL GRANT host name。

批量执行初始化 SQL 语句

集群在初始化过程还可以自动执行 `initSql` 中的 SQL 语句用于初始化，该功能可以用于默认给集群创建一些 database 或者 table，并且执行一些用户权限管理类的操作。例如如下设置会在集群创建完成后自动创建名为 `app` 的 database，并且赋予 `developer` 账号对 `app` 的所有管理权限：

```
spec:
...
initSql: |-
    CREATE DATABASE app;
    GRANT ALL PRIVILEGES ON app.* TO 'developer'@'%';
```

> **注意：**

> 目前没有对 initSql 做校验，尽管也可以在 initSql 里面创建账户和设置密码，但这种方式会将密码以明文形式存到 initializer Job 对象上，不建议这么做。

测试环境示例 `tidb-initializer.yaml`

```yaml
---
apiVersion: pingcap.com/v1alpha1
kind: TidbInitializer
metadata:
  name: demo-init
  namespace: tidb-cluster
spec:
  image: tnir/mysqlclient
  # imagePullPolicy: IfNotPresent
  cluster:
    namespace: tidb-cluster
    name: tidb-cluster
  # initSql: |-
    # create database app;
  # initSqlConfigMap: tidb-initsql
  passwordSecret: tidb-secret
  # permitHost: 172.6.5.8
  # resources:
  #   limits:
  #     cpu: 1000m
  #     memory: 500Mi
  #   requests:
  #     cpu: 100m
  #     memory: 50Mi
  # timezone: "Asia/Shanghai"
```

1.5.2 执行初始化

```
kubectl apply -f ${cluster_name}/tidb-initializer.yaml --
namespace=${namespace}

# 测试环境示例
kubectl apply -f tidb-cluster/tidb-initializer.yaml --namespace=tidb-cluster
```

以上命令会自动创建一个初始化的 Job，该 Job 会尝试利用提供的 secret 给 root 账号创建初始密码，并且创建其它账号和密码（如果指定了的话）。初始化完成后 Pod 状态会变成 Completed，之后通过 MySQL 客户端登录时需要指定这里设置的密码。

1.6 访问 TiDB 集群

由于 TiDB 支持 MySQL 传输协议及其绝大多数的语法，因此可以直接使用 `mysql` 命令行工具连接 TiDB 进行操作。以下说明连接 TiDB 集群的步骤。

1.6.1 安装 `mysql` 命令行工具

要连接到 TiDB，您需要在使用 `kubectl` 的主机上安装与 MySQL 兼容的命令行客户端。可以安装 MySQL Server，MariaDB Server，Percona Server 的 mysql 可执行文件，也可以从操作系统软件仓库中安装。

1.6.2 获取 TiDB Service 信息

可以通过如下命令获取 TiDB Service 信息：

```
kubectl get svc ${serviceName} -n ${namespace}
```

示例：

```
# kubectl get svc -n tidb-cluster
NAME                       TYPE       CLUSTER-IP     EXTERNAL-IP  PORT(S)
                     AGE
tidb-cluster-discovery     ClusterIP  10.101.3.165   <none>
10261/TCP,10262/TCP              137m
tidb-cluster-pd            ClusterIP  10.103.99.236  <none>       2379/TCP
                     137m
tidb-cluster-pd-peer       ClusterIP  None           <none>       2380/TCP
                     137m
tidb-cluster-tidb          NodePort   10.98.76.144   <none>
4000:30430/TCP,10080:31986/TCP  134m
tidb-cluster-tidb-peer     ClusterIP  None           <none>       10080/TCP
                     134m
tidb-cluster-tiflash-peer ClusterIP  None           <none>
3930/TCP,20170/TCP               136m
tidb-cluster-tikv-peer     ClusterIP  None           <none>       20160/TCP
                     136m
```

示例描述了 `tidb-cluster` namespace 下 的服务信息，`tidb-cluster-tidb` 服务 类型为 `NodePort`，ClusterIP 为 `10.98.76.144`，ServicePort 为 `4000` 和 `10080`，对应的 NodePort 分别为 `30430` 和 `31986`。

ClusterIP

`ClusterIP` 是通过集群的内部 IP 暴露服务，选择该类型的服务时，只能在集群内部访问，可以通过如下方式访问：

- ClusterIP + ServicePort
- Service 域名 (`${serviceName}.${namespace}`) + ServicePort

NodePort

在没有 LoadBalancer 时，可选择通过 NodePort 暴露。NodePort 是通过节点的 IP 和静态端口暴露服务。通过请求 `NodeIP + NodePort`，可以从集群的外部访问一个 NodePort 服务。

查看 Service 分配的 Node Port，可通过获取 TiDB 的 Service 对象来获知：

```
kubectl -n ${namespace} get svc ${cluster_name}-tidb -ojsonpath="
{.spec.ports[?(@.name=='mysql-client')].nodePort}{'\n'}"

# 测试环境示例，输出30430
kubectl -n tidb-cluster get svc tidb-cluster-tidb -ojsonpath="{.spec.ports[?
(@.name=='mysql-client')].nodePort}{'\n'}"
```

查看可通过哪些节点的 IP 访问 TiDB 服务，有两种情况：

- `externalTrafficPolicy` 为 `Cluster` 时，所有节点 IP 均可
- `externalTrafficPolicy` 为 `Local` 时，可通过以下命令获取指定集群的 TiDB 实例所在的节点

```
kubectl -n ${namespace} get pods -l "app.kubernetes.io/component=tidb,app.kubernetes.io/instance=${cluster_name}" -ojsonpath="{range .items[*]}{.spec.nodeName}{'\n'}{end}"

# 测试环境示例，externalTrafficPolicy 为 Local，输出：
# vm-0-4-centos
# vm-0-3-centos
kubectl -n tidb-cluster get pods -l "app.kubernetes.io/component=tidb,app.kubernetes.io/instance=tidb-cluster" -ojsonpath="{range .items[*]}{.spec.nodeName}{'\n'}{end}"
```

LoadBalancer

若运行在有 LoadBalancer 的环境，比如 GCP/AWS 平台，建议使用云平台的 LoadBalancer 特性。

1.6.3 连接 TiDB 服务

> **注意：**
>
> 当使用 MySQL Client 8.0 访问 TiDB 服务（TiDB 版本 < v4.0.7）时，如果用户账户有配置密码，必须显式指定 `--default-auth=mysql_native_password` 参数，因为 `mysql_native_password` 不再是默认的插件。

```
mysql -h 127.0.0.1 -P 30430 -u root

# 如果执行初始化设置了密码，输入以下命令后输入密码即可
mysql -h 127.0.0.1 -P 30430 -u root -p
```

期望输出：

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 76
Server version: 5.7.25-TiDB-v4.0.0 MySQL Community Server (Apache License 2.0)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

1.7 销毁集群

```
# 删除所有集群组件实例的所有 pod，但不删除PVC，需手动删除
kubectl delete tc tidb-cluster -n tidb-cluster
```

1.8 导入集群数据

使用 TiDB Lightning 导入集群数据。MySQL数据源准备请见 **2.9.1 使用 Dumpling导出数据** 小节。

1.8.1 配置 TiDB Lightning

使用如下命令获得 TiDB Lightning 的默认配置：

```
helm inspect values pingcap/tidb-lightning --version=${chart_version} > tidb-lightning-values.yaml

# 测试环境示例
helm inspect values pingcap/tidb-lightning --version=v1.2.4 > tidb-lightning-values.yaml
```

> **注意：**
>
> `${chart_version}` 在后续文档中代表 chart 版本，例如 `v1.2.4`

tidb-lightning Helm chart 支持恢复本地或远程的备份数据。

本地模式

本地模式要求备份工具导出的备份数据位于其中一个 Kubernetes 节点上。要启用该模式，你需要将 `dataSource.local.nodeName` 设置为该节点名称，将 `dataSource.local.hostPath` 设置为备份数据目录路径，该路径中需要包含名为 `metadata` 的文件。

测试环境配置示例`tidb-lightning-values.yaml`：

```
# Default values for tidb-lightning.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

# timezone is the default system timzone
timezone: Asia/Shanghai

image: pingcap/tidb-lightning:v5.2.1
imagePullPolicy: IfNotPresent
# imagePullSecrets: []

service:
  type: NodePort

# failFast causes the lightning pod fails when any error happens.
# when disabled, the lightning pod will keep running when error happens to allow manual intervention, users have to check logs to see the job status.
failFast: true

dataSource:
  # for `local` source, the `nodeName` should be the label value of `kubernetes.io/hostname`.
  local:
     nodeName: vm-0-3-centos
     hostPath: /workspace/Koinophobia/TiDB/data/my_database

targetTidbCluster:
  name: tidb-cluster
  # namespace is the target tidb cluster namespace, can be omitted if the lightning is deployed in the same namespace of the target tidb cluster
  namespace: "tidb-cluster"
  user: root
```

```yaml
  # If the `secretName` and `secretUserKey` are set,
  # the `user` will be ignored and the user in the
  # `secretName` will be used by lightning.
  # If the `secretName` and `secretPwdKey` are set, the
  # password in the `secretName` will be used by lightning.
  #secretName: ""
  #secretUserKey: user
  #secretPwdKey: password


tlsCluster: {}
  # enabled: true

tlsClient: {}
  # enabled: true
  # tlsClientSecretName: ${targetTidbCluster.name}-tidb-client-secret

resources: {}
  # limits:
  #  cpu: 16000m
  #  memory: 8Gi
  # requests:
  #  cpu: 16000m
  #  memory: 8Gi

nodeSelector: {}

annotations: {}

tolerations: []
affinity: {}

# The delivery backend used to import data (valid options include
`importer`, `local` and `tidb`).
# If set to `local`, then the following `sortedKV` should be set.
backend: local

# For `local` backend, an extra PV is needed for local KV sorting.
sortedKV:
 storageClassName: local-path
 storage: 3Gi

config: |
  [lightning]
  level = "info"
  file = "-"
  [checkpoint]
  enable = true
  driver = "file"
  dsn = "CHECKPOINT_USE_DATA_DIR/tidb_lightning_checkpoint.pb"
  keep-after-success = false

```

1.8.2 部署 TiDB Lightning

部署 TiDB Lightning 的方式根据不同的权限授予方式及存储方式，有不同的情况。

对于本地模式、Ad hoc 模式、远程模式（需要是符合以下三个条件之一的远程模式：使用 Amazon S3 AccessKey 和 SecretKey 权限授予方式、使用 Ceph 作为存储后端、使用 GCS 作为存储后端），运行以下命令部署 TiDB Lightning：

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${namespace} --set failFast=true -f tidb-lightning-values.yaml --version=${chart_version}

# 测试环境示例
helm install tidb-lightning pingcap/tidb-lightning --namespace=tidb-cluster --set failFast=true -f ./tidb-cluster/tidb-lightning-values.yaml --version=v1.2.4
```

预期输出：

```
NAME: tidb-lightning
LAST DEPLOYED: Mon Nov 22 16:38:28 2021
NAMESPACE: tidb-cluster
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Check tidb-lightning status
  kubectl get job -n tidb-cluster -l app.kubernetes.io/name=tidb-lightning-tidb-lightning
  kubectl get po -n tidb-cluster -l app.kubernetes.io/name=tidb-lightning-tidb-lightning
2. Check tidb-lightning logs
  kubectl logs -n tidb-cluster -l app.kubernetes.io/name=tidb-lightning-tidb-lightning
3. View tidb-lightning status page
  kubectl port-forward -n tidb-cluster svc/tidb-lightning-tidb-lightning 8289:8289
  View http://localhost:8289 in web browser
```

另外可用 `kubectl describe -n tidb-cluster pod ${tidb-lightning-pod-name}` 命令查看启动情况。

1.8.3 销毁 TiDB Lightning

目前，TiDB Lightning 只能在线下恢复数据。当恢复过程结束、TiDB 集群需要向外部应用提供服务时，可以销毁 TiDB Lightning 以节省开支。

删除 tidb-lightning 的方法：

- 运行 `helm uninstall ${release_name} -n ${namespace}`。

```
# 测试环境示例
helm uninstall tidb-lightning -n tidb-cluster
```

附：

```
# 文件拷本命令
scp -r /workspace/Koinophobia/TiDB/data
root@172.31.215.187:/workspace/Koinophobia/TiDB
```

2. 本地部署 TiDB 集群

TiUP 是 TiDB 4.0 版本引入的集群运维工具，TiUP cluster 是 TiUP 提供的使用 Golang 编写的集群管理组件，通过 TiUP cluster 组件就可以进行日常的运维工作，包括部署、启动、关闭、销毁、弹性扩缩容、升级 TiDB 集群，以及管理 TiDB 集群参数。

2.1 软硬件环境需求及前置检查

软硬件环境需求

环境与系统配置检查

2.2 在中控机上安装 TiUP 组件

在中控机上安装 TiUP 组件有两种方式：在线部署和离线部署。

方式一：在线部署 TiUP 组件（推荐）

使用普通用户登录中控机，以 `tidb` 用户为例，后续安装 TiUP 及集群管理操作均通过该用户完成：

1. 执行如下命令安装 TiUP 工具：

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-
mirrors.pingcap.com/install.sh | sh
# 此命令会提示.bash_profile目录信息
```

2. 按如下步骤设置 TiUP 环境变量：

   重新声明全局环境变量：

```
# .bash_profile具体位置在命令行会有提示
source .bash_profile
```

   确认 TiUP 工具是否安装：

```
which tiup
```

3. 安装 TiUP cluster 组件

```
tiup cluster
```

4. 如果已经安装，则更新 TiUP cluster 组件至最新版本：

```
tiup update --self && tiup update cluster
```

   预期输出 `“Update successfully!”` 字样。

5. 验证当前 TiUP cluster 版本信息。执行如下命令查看 TiUP cluster 组件版本：

```
tiup --binary cluster
```

方式二：离线部署 TiUP 组件

离线部署 TiUP 组件的操作步骤如下。

准备 TiUP 离线组件包

方式一：在官方下载页面选择对应版本的 TiDB server 离线镜像包（包含 TiUP 离线组件包）。

方式二：使用 `tiup mirror clone` 命令手动打包离线组件包。步骤如下：

1. 在在线环境中安装 TiUP 包管理器工具
   1). 执行如下命令安装 TiUP 工具：

   ```
   curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh
   ```

   2). 重新声明全局环境变量：

   ```
   source .bash_profile
   ```

   3). 确认 TiUP 工具是否安装：

   ```
   which tiup
   ```

2. 使用 TiUP 制作离线镜像
   1). 在一台和外网相通的机器上拉取需要的组件：

   ```
   tiup mirror clone tidb-community-server-${version}-linux-amd64 ${version} --os=linux --arch=amd64
   ```

   该命令会在当前目录下创建一个名叫 `tidb-community-server-${version}-linux-amd64` 的目录，里面包含 TiUP 管理的组件包。

   2). 通过 tar 命令将该组件包打包然后发送到隔离环境的中控机：

   ```
   tar czvf tidb-community-server-${version}-linux-amd64.tar.gz tidb-community-server-${version}-linux-amd64
   ```

   此时，`tidb-community-server-${version}-linux-amd64.tar.gz` 就是一个独立的离线环境包。

3. 自定义制作的离线镜像，或调整已有离线镜像中的内容

   如果从官网下载的离线镜像不满足你的具体需求，或者希望对已有的离线镜像内容进行调整，例如增加某个组件的新版本等，可以采取以下步骤进行操作：

   1). 在制作离线镜像时，可通过参数指定具体的组件和版本等信息，获得不完整的离线镜像。例如，要制作一个只包括 v1.5.2 版本 TiUP 和 TiUP Cluster 的离线镜像，可执行如下命令：

   ```
   tiup mirror clone tiup-custom-mirror-v1.5.2 --tiup v1.5.2 --cluster v1.5.2
   ```

   如果只需要某一特定平台的组件，也可以通过 `--os` 和 `--arch` 参数来指定。

2). 参考“使用 TiUP 制作离线镜像”第 2 步的方式，将此不完整的离线镜像传输到隔离环境的中控机。

3). 在隔离环境的中控机上，查看当前使用的离线镜像路径。较新版本的 TiUP 可以直接通过命令获取当前的镜像地址：

```
tiup mirror show
```

以上命令如果提示 `show` 命令不存在，可能当前使用的是较老版本的 TiUP。此时可以通过查看 `$HOME/.tiup/tiup.toml` 获得正在使用的镜像地址。将此镜像地址记录下来，后续步骤中将以变量 `${base_mirror}` 指代此镜像地址。

4). 将不完整的离线镜像合并到已有的离线镜像中：

首先将当前离线镜像中的 `keys` 目录复制到 `$HOME/.tiup` 目录中：

```
cp -r ${base_mirror}/keys $HOME/.tiup/
```

然后使用 TiUP 命令将不完整的离线镜像合并到当前使用的镜像中：

```
tiup mirror merge tiup-custom-mirror-v1.5.2
```

上述步骤完成后，通过 `tiup list` 命令检查执行结果。在本文例子中，使用 `tiup list tiup` 和 `tiup list cluster` 均应能看到对应组件的 `v1.5.2` 版本出现在结果中。

部署离线环境 TiUP 组件

将离线包发送到目标集群的中控机后，执行以下命令安装 TiUP 组件：

```
tar xzvf tidb-community-server-${version}-linux-amd64.tar.gz && \
sh tidb-community-server-${version}-linux-amd64/local_install.sh && \
source /home/tidb/.bash_profile
```

`local_install.sh` 脚本会自动执行 `tiup mirror set tidb-community-server-${version}-linux-amd64` 命令将当前镜像地址设置为 `tidb-community-server-${version}-linux-amd64`。

若需将镜像切换到其他目录，可以通过手动执行 `tiup mirror set <mirror-dir>` 进行切换。如果需要切换到在线环境，可执行 `tiup mirror set https://tiup-mirrors.pingcap.com`。

2.3 初始化集群拓扑文件

请根据不同的集群拓扑，编辑 TiUP 所需的集群初始化配置文件。

集群初始化配置文件可以通过 TiUP 工具在中控机上面创建 YAML 格式配置文件，例如 `topology.yaml`：

```
tiup cluster template > topology.yaml
```

> **注意：**
>
> 混合部署场景也可以使用 `tiup cluster template --full > topology.yaml` 生成的建议拓扑模板，跨机房部署场景可以使用 `tiup cluster template --multi-dc > topology.yaml` 生成的建议拓扑模板。

执行 `vi topology.yaml`，查看配置文件的内容：

```yaml
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/tidb-deploy"
  data_dir: "/tidb-data"
server_configs: {}
pd_servers:
  - host: 10.0.1.4
  - host: 10.0.1.5
  - host: 10.0.1.6
tidb_servers:
  - host: 10.0.1.7
  - host: 10.0.1.8
  - host: 10.0.1.9
tikv_servers:
  - host: 10.0.1.1
  - host: 10.0.1.2
  - host: 10.0.1.3
monitoring_servers:
  - host: 10.0.1.4
grafana_servers:
  - host: 10.0.1.4
alertmanager_servers:
  - host: 10.0.1.4
```

这里举出常见的 6 种场景，请根据链接中的拓扑说明，以及给出的配置文件模板，修改配置文件 `topology.yaml`。如果有其他组合场景的需求，请根据标准模板自行调整。

- 最小拓扑架构

  最基本的集群拓扑，包括 tidb-server、tikv-server、pd-server，适合 OLTP 业务。

- 增加 TiFlash 拓扑架构

  包含最小拓扑的基础上，同时部署 TiFlash。TiFlash 是列式的存储引擎，已经逐步成为集群拓扑的标配。适合 Real-Time HTAP 业务。

- 增加 TiCDC 拓扑架构

  包含最小拓扑的基础上，同时部署 TiCDC。TiCDC 是 4.0 版本开始支持的 TiDB 增量数据同步工具，支持多种下游（TiDB/MySQL/MQ）。相比于 TiDB Binlog，TiCDC 有延迟更低、天然高可用等优点。在部署完成后，需要启动 TiCDC，通过 `cdc cli` 创建同步任务。

- 增加 TiDB Binlog 拓扑架构

  包含最小拓扑的基础上，同时部署 TiDB Binlog。TiDB Binlog 是目前广泛使用的增量同步组件，可提供准实时备份和同步功能。

- 增加 TiSpark 拓扑架构

  包含最小拓扑的基础上，同时部署 TiSpark 组件。TiSpark 是 PingCAP 为解决用户复杂 OLAP 需求而推出的产品。TiUP cluster 组件对 TiSpark 的支持目前为实验性特性。

- 混合部署拓扑架构

  适用于单台机器，混合部署多个实例的情况，也包括单机多实例，需要额外增加目录、端口、资源配比、label 等配置。

- 跨机房部署拓扑架构

  以典型的 `两地三中心` 架构为例，介绍跨机房部署架构，以及需要注意的关键设置。

> **注意：**

- 对于需要全局生效的参数，请在配置文件中 `server_configs` 的对应组件下配置。
- 对于需要某个节点生效的参数，请在具体节点的 `config` 中配置。
- 配置的层次结构使用 `.` 表示。如：`log.slow-threshold`。更多格式参考 TiUP 配置参数模版。
- 更多参数说明，请参考 TiDB `config.toml.example`、TiKV `config.toml.example`、PD `config.toml.example` 和 TiFlash 配置参数。

开发环境拓文件

包括TiDB、PD、TiKV、TiFlash以及监控组件，仅供参考：

```
# # Global variables are applied to all deployments and used as the default value of
# # the deployments if a specific deployment value is missing.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/workspace/tidb/tidb-deploy"
  data_dir: "/workspace/tidb/tidb-data"

# # Monitored variables are applied to all the machines.
monitored:
  node_exporter_port: 9100
  blackbox_exporter_port: 9115
  # deploy_dir: "/tidb-deploy/monitored-9100"
  # data_dir: "/tidb-data/monitored-9100"
  # log_dir: "/tidb-deploy/monitored-9100/log"

# # Server configs are used to specify the runtime configuration of TiDB components.
# # All configuration items can be found in TiDB docs:
# # - TiDB: https://pingcap.com/docs/stable/reference/configuration/tidb-server/configuration-file/
# # - TiKV: https://pingcap.com/docs/stable/reference/configuration/tikv-server/configuration-file/
# # - PD: https://pingcap.com/docs/stable/reference/configuration/pd-server/configuration-file/
# # All configuration items use points to represent the hierarchy, e.g:
# #   readpool.storage.use-unified-pool
# #
# # You can overwrite this configuration via the instance-level `config` field.

server_configs:
  tidb:
    log.slow-threshold: 300
    binlog.enable: false
    binlog.ignore-error: false
  tikv:
    # server.grpc-concurrency: 4
    # raftstore.apply-pool-size: 2
    # raftstore.store-pool-size: 2
    # rocksdb.max-sub-compactions: 1
    # storage.block-cache.capacity: "16GB"
    # readpool.unified.max-thread-count: 12
    readpool.storage.use-unified-pool: false
    readpool.coprocessor.use-unified-pool: true
```

```
  pd:
    replication.enable-placement-rules: true
    schedule.leader-schedule-limit: 4
    schedule.region-schedule-limit: 2048
    schedule.replica-schedule-limit: 64
  tiflash:
    # Maximum memory usage for processing a single query. Zero means
unlimited.
    profiles.default.max_memory_usage: 0
    # Maximum memory usage for processing all concurrently running queries
on the server. Zero means unlimited.
    profiles.default.max_memory_usage_for_all_queries: 0

# 节点:
# vm-0-1-centos   172.31.215.191   worker1  TiDB+ TiKV + Monitor
# vm-0-2-centos   172.31.215.188   worker2  PD + TiKV
# vm-0-3-centos   172.31.215.187   worker3  TiKV
# vm-0-4-centos   172.31.215.189   worker4  TiDB + TiFlash
# vm-0-5-centos   172.31.215.192   worker5  TiDB + PD + TiFlash

pd_servers:
  - host: 172.31.215.192
    # ssh_port: 22
    # name: "pd-1"
    client_port: 2479
    peer_port: 2480
    # deploy_dir: "/tidb-deploy/pd-2379"
    # data_dir: "/tidb-data/pd-2379"
    # log_dir: "/tidb-deploy/pd-2379/log"
    # numa_node: "0,1"
    # # The following configs are used to overwrite the `server_configs.pd`
values.
    # config:
    #   schedule.max-merge-region-size: 20
    #   schedule.max-merge-region-keys: 200000
  - host: 172.31.215.188
    client_port: 2479
    peer_port: 2480

tidb_servers:
  - host: 172.31.215.191
    # ssh_port: 22
    # port: 4000
    # status_port: 10080
    # deploy_dir: "/tidb-deploy/tidb-4000"
    # log_dir: "/tidb-deploy/tidb-4000/log"
    # numa_node: "0,1"
    # # The following configs are used to overwrite the
`server_configs.tidb` values.
    # config:
    #   log.slow-query-file: tidb-slow-overwrited.log
  - host: 172.31.215.189
  - host: 172.31.215.192

tikv_servers:
  - host: 172.31.215.191
    # ssh_port: 22
    # port: 20160
```

```
    # status_port: 20180
    # deploy_dir: "/tidb-deploy/tikv-20160"
    # data_dir: "/tidb-data/tikv-20160"
    # log_dir: "/tidb-deploy/tikv-20160/log"
    # numa_node: "0,1"
    # # The following configs are used to overwrite the
`server_configs.tikv` values.
    # config:
    #   server.grpc-concurrency: 4
    #   server.labels: { zone: "zone1", dc: "dc1", host: "host1" }
  - host: 172.31.215.188
  - host: 172.31.215.187

tiflash_servers:
  - host: 172.31.215.189
    # ssh_port: 22
    # tcp_port: 9000
    # http_port: 8123
    # flash_service_port: 3930
    # flash_proxy_port: 20170
    # flash_proxy_status_port: 20292
    # metrics_port: 8234
    # deploy_dir: /tidb-deploy/tiflash-9000
    ## The `data_dir` will be overwritten if you define `storage.main.dir`
configurations in the `config` section.
    # data_dir: /tidb-data/tiflash-9000
    # numa_node: "0,1"
    # # The following configs are used to overwrite the
`server_configs.tiflash` values.
    # config:
    #   logger.level: "info"
    #   ## Multi-disk deployment introduced in v4.0.9
    #   ## Check https://docs.pingcap.com/tidb/stable/tiflash-
configuration#multi-disk-deployment for more details.
    #   ## Example1:
    #   # storage.main.dir: [ "/nvme_ssd0_512/tiflash",
"/nvme_ssd1_512/tiflash" ]
    #   # storage.main.capacity = [ 536870912000, 536870912000 ]
    #   ## Example2:
    #   # storage.main.dir: [ "/sata_ssd0_512/tiflash",
"/sata_ssd1_512/tiflash", "/sata_ssd2_512/tiflash" ]
    #   # storage.latest.dir: [ "/nvme_ssd0_150/tiflash" ]
    #   # storage.main.capacity = [ 536870912000, 536870912000 ]
    #   # storage.latest.capacity = [ 161061273600 ]
    # learner_config:
    #   log-level: "info"
  - host: 172.31.215.192

monitoring_servers:
  - host: 172.31.215.191
    # ssh_port: 22
    # port: 9090
    # deploy_dir: "/tidb-deploy/prometheus-8249"
    # data_dir: "/tidb-data/prometheus-8249"
    # log_dir: "/tidb-deploy/prometheus-8249/log"

grafana_servers:
  - host: 172.31.215.191
```

```
    # port: 3000
    # deploy_dir: /tidb-deploy/grafana-3000

alertmanager_servers:
  - host: 172.31.215.191
    # ssh_port: 22
    # web_port: 9093
    # cluster_port: 9094
    # deploy_dir: "/tidb-deploy/alertmanager-9093"
    # data_dir: "/tidb-data/alertmanager-9093"
    # log_dir: "/tidb-deploy/alertmanager-9093/log"
```

2.4 执行部署命令

> **注意：**
>
> 通过 TiUP 进行集群部署可以使用密钥或者交互密码方式来进行安全认证：
>
> - 如果是密钥方式，可以通过 `-i` 或者 `--identity_file` 来指定密钥的路径；
> - 如果是密码方式，可以通过 `-p` 进入密码交互窗口；
> - 如果已经配置免密登录目标机，则不需填写认证。
>
> 一般情况下 TiUP 会在目标机器上创建 `topology.yaml` 中约定的用户和组，以下情况例外：
>
> - `topology.yaml` 中设置的用户名在目标机器上已存在。
> - 在命令行上使用了参数 `--skip-create-user` 明确指定跳过创建用户的步骤。

执行 deploy 命令前，先使用 `check` 及 `check --apply` 命令，检查和自动修复集群存在的潜在风险：

```
tiup cluster check ./topology.yaml --user root [-p] [-i
/home/root/.ssh/gcp_rsa]
tiup cluster check ./topology.yaml --apply --user root [-p] [-i
/home/root/.ssh/gcp_rsa]
```

然后执行 `deploy` 命令部署 TiDB 集群：

```
tiup cluster deploy tidb-test v5.2.2 ./topology.yaml --user root [-p] [-i
/home/root/.ssh/gcp_rsa]
```

以上部署命令中：

- 通过 TiUP cluster 部署的集群名称为 `tidb-test`
- 可以通过执行 `tiup list tidb` 来查看 TiUP 支持的最新可用版本，后续内容以版本 `v5.2.2` 为例
- 初始化配置文件为 `topology.yaml`
- --user root：通过 root 用户登录到目标主机完成集群部署，该用户需要有 ssh 到目标机器的权限，并且在目标机器有 sudo 权限。也可以用其他有 ssh 和 sudo 权限的用户完成部署。
- [-i] 及 [-p]：非必选项，如果已经配置免密登录目标机，则不需填写。否则选择其一即可，[-i] 为可登录到目标机的 root 用户（或 --user 指定的其他用户）的私钥，也可使用 [-p] 交互式输入该用户的密码
- 如果需要指定在目标机创建的用户组名，可以参考这个例子。

预期日志结尾输出会有 `Deployed cluster` tidb-test `successfully` 关键词，表示部署成功。

2.5 查看 TiUP 管理的集群情况

```
tiup cluster list
```

TiUP 支持管理多个 TiDB 集群，该命令会输出当前通过 TiUP cluster 管理的所有集群信息，包括集群名称、部署用户、版本、密钥信息等：

```
Starting /home/tidb/.tiup/components/cluster/v1.5.0/cluster list
Name              User  Version        Path
                                       PrivateKey
----              ----  -------        ----
                                       ----------
tidb-test         tidb  v5.2.2
/home/tidb/.tiup/storage/cluster/clusters/tidb-test
/home/tidb/.tiup/storage/cluster/clusters/tidb-test/ssh/id_rsa
```

2.6 检查部署的 TiDB 集群情况

例如，执行如下命令检查 `tidb-test` 集群情况：

```
tiup cluster display tidb-test
```

预期输出包括 `tidb-test` 集群中实例 ID、角色、主机、监听端口和状态（由于还未启动，所以状态为 Down/inactive）、目录信息。

2.7 启动集群

```
tiup cluster start tidb-test
```

预期结果输出 `Started cluster` tidb-test `successfully` 标志启动成功。

2.8 验证集群运行状态

操作步骤见验证集群运行状态。

2.9 数据迁移

2.9.1 使用 Dumpling 导出数据

使用数据导出工具 Dumpling，你可以把存储在 TiDB 或 MySQL 中的数据导出为 SQL 或 CSV 格式，用于逻辑全量备份。

可以通过下列任意方式获取 Dumpling：

- TiUP 执行 `tiup install dumpling` 命令。获取后，使用 `tiup dumpling ...` 命令运行 Dumpling。
- 下载包含 Dumpling 的 tidb-toolkit 安装包。

从 TiDB/MySQL 导出数据

需要的权限

- SELECT
- RELOAD
- LOCK TABLES
- REPLICATION CLIENT
- PROCESS

导出为 SQL 文件

假设在 `127.0.0.1:4000` 有一个 TiDB 实例，并且这个 TiDB 实例中有无密码的 root 用户。

Dumpling 默认导出数据格式为 SQL 文件。也可以通过设置 `--filetype sql` 导出数据到 SQL 文件：

```
dumpling \
  -u root \
  -P 4000 \
  -h 127.0.0.1 \
  --filetype sql \
  -t 8 \
  -o /tmp/test \
  -r 200000 \
  -F 256MiB
```

以上命令中：

- `-h`、`-P`、`-u` 分别代表地址、端口、用户。如果需要密码验证，可以使用 `-p $YOUR_SECRET_PASSWORD` 将密码传给 Dumpling。
- `-o` 用于选择存储导出文件的目录，支持本地文件路径或外部存储 URL 格式。
- `-t` 用于指定导出的线程数。增加线程数会增加 Dumpling 并发度提高导出速度，但也会加大数据库内存消耗，因此不宜设置过大。
- `-r` 用于指定单个文件的最大行数，指定该参数后 Dumpling 会开启表内并发加速导出，同时减少内存使用。
- `-F` 选项用于指定单个文件的最大大小，单位为 `MiB`，可接受类似 `5GiB` 或 `8KB` 的输入。如果你想使用 TiDB Lightning 将该文件加载到 TiDB 实例中，建议将 `-F` 选项的值保持在 256 MiB 或以下。

> **注意：**
>
> 如果导出的单表大小超过 10 GB，**强烈建议**使用 `-r` 和 `-F` 参数。

导出为 CSV 文件

假如导出数据的格式是 CSV（使用 `--filetype csv` 即可导出 CSV 文件），还可以使用 `--sql <SQL>` 导出指定 SQL 选择出来的记录，例如，导出 `test.sbtest1` 中所有 `id < 100` 的记录：

```
./dumpling \
  -u root \
  -P 4000 \
  -h 127.0.0.1 \
  -o /tmp/test \
  --filetype csv \
  --sql 'select * from `test`.`sbtest1` where id < 100'
```

> **注意：**
>
> - `--sql` 选项仅仅可用于导出 CSV 的场景。

> - 该命令将在要导出的所有表上执行 `select * from <table-name> where id < 100` 语句。如果部分表没有指定的字段，那么导出会失败。
> - Dumpling 导出不区分`字符串`与`关键字`。如果导入的数据是 Boolean 类型的 `true` 和 `false`，导出时会被转换为 `1` 和 `0` 。

输出文件格式

- `metadata`：此文件包含导出的起始时间，以及 master binary log 的位置。

```
cat metadata
Started dump at: 2020-11-10 10:40:19
SHOW MASTER STATUS:
        Log: tidb-binlog
        Pos: 420747102018863124

Finished dump at: 2020-11-10 10:40:20
```

`{schema}-schema-create.sql`：创建 schema 的 SQL 文件。

```
cat test-schema-create.sql
CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
```

`{schema}.{table}-schema.sql`：创建 table 的 SQL 文件

```
cat test.t1-schema.sql
CREATE TABLE `t1` (
  `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

`{schema}.{table}.{0001}.{sql|csv}`：数据源文件

```
cat test.t1.0.sql
/*!40101 SET NAMES binary*/;
INSERT INTO `t1` VALUES
(1);
```

`*-schema-view.sql`、`*-schema-trigger.sql`、`*-schema-post.sql`：其他导出文件

2.9.2 使用 TiDB Lightning 导入数据：

TiDB Lightning 是一个将全量数据高速导入到 TiDB 集群的工具，目前支持 SQL 或 CSV 输出格式的数据源。

准备全量备份数据

使用 dumpling 从 MySQL 导出数据，如下：

```
./bin/dumpling -h 127.0.0.1 -P 3306 -u root -t 16 -F 256MB -B test -f
'test.t[12]' -o /data/my_database/
```

其中：

- `-B test`：从 `test` 数据库导出。
- `-f test.t[12]`：只导出 `test.t1` 和 `test.t2` 这两个表。

- `-t 16`：使用 16 个线程导出数据。
- `-F 256MB`：将每张表切分成多个文件，每个文件大小约为 256 MB。

这样全量备份数据就导出到了 `/data/my_database` 目录中。

部署 TiDB Lightning

下载 TiDB Lightning 安装包

通过以下链接获取 TiDB Lightning 安装包（TiDB Lightning 完全兼容较低版本的 TiDB 集群，建议选择最新稳定版本）：

- **v5.0.0**: tidb-toolkit-v5.0.0-linux-amd64.tar.gz

启动 `tidb-lightning`

1. 将安装包里的 `bin/tidb-lightning` 及 `bin/tidb-lightning-ctl` 上传至部署 TiDB Lightning 的服务器。
2. 将数据源也上传到同样的服务器。
3. 配置 `tidb-lightning.toml`。

```
[lightning]
# 日志
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# 选择使用的 local 后端
backend = "local"
# 设置排序的键值对的临时存放地址，目标路径需要是一个空目录
sorted-kv-dir = "/mnt/ssd/sorted-kv-dir"

[mydumper]
# 源数据目录。
data-source-dir = "/data/my_datasource/"

# 配置通配符规则，默认规则会过滤 mysql、sys、INFORMATION_SCHEMA、PERFORMANCE_SCHEMA、METRICS_SCHEMA、INSPECTION_SCHEMA 系统数据库下的所有表
# 若不配置该项，导入系统表时会出现“找不到 schema”的异常
filter = ['*.*', '!mysql.*', '!sys.*', '!INFORMATION_SCHEMA.*', '!PERFORMANCE_SCHEMA.*', '!METRICS_SCHEMA.*', '!INSPECTION_SCHEMA.*']
[tidb]
# 目标集群的信息
host = "172.16.31.2"
port = 4000
user = "root"
password = "rootroot"
# 表架构信息在从 TiDB 的“状态端口”获取。
status-port = 10080
# 集群 pd 的地址
pd-addr = "172.16.31.3:2379"
```

配置合适的参数运行 `tidb-lightning`。如果直接在命令行中用 `nohup` 启动程序，可能会因为 SIGHUP 信号而退出，建议把 `nohup` 放到脚本里面，如：

```
#!/bin/bash
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

检查数据

导入完毕后，TiDB Lightning 会自动退出。若导入成功，日志的最后一行会显示 `tidb lightning exit`。

如果出错，请参见 TiDB Lightning 常见问题。