9.TARS Kubernetes 部署

Kubernetes

因为TARS本身提供了很多集群与微服务治理的工具，而在TARS研发之初是没有考虑跟Kubernetes（以下简称 K8s）相结合的，所以我们需要在部署的过程中想一些办法。

K8s的部署，我们可以使用 ansible 工具 https://github.com/gjmzj/kubeasz

这里我用了5台服务器，这些IP后面的介绍里面会用到，可以根据自己的规划来修改：

- 172.16.200.151 - Deploy Server，用来 **运行 ansible 脚本**
- 172.16.200.152 - K8s Master，主节点，**执行 kubectl 命令**
- 172.16.200.153 - K8s Worker Node，负载节点
- 172.16.200.154 - K8s Worker Node，负载节点
- 172.16.200.155 - NTP & NFS Server

下面内容里的 YAML 文件在主节点运行 `kubectl create -f XXX.yaml` 即可执行。

NFS

NFS服务器（在 172.16.200.155 上执行）：

```
yum install nfs-utils -y

mkdir /home/nfsshare
chmod -R 755 /home/nfsshare
chown -R nfsnobody:nfsnobody /home/nfsshare

echo '/home/nfsshare    172.16.200.0/24
(rw,sync,no_root_squash,no_all_squash)' >>  /etc/exports

systemctl enable rpcbind
systemctl enable nfs-server
systemctl enable nfs-lock
systemctl enable nfs-idmap
```

```
systemctl start rpcbind
systemctl start nfs-server
systemctl start nfs-lock
systemctl start nfs-idmap

firewall-cmd --permanent --zone=public --add-service=nfs
firewall-cmd --permanent --zone=public --add-service=mountd
firewall-cmd --permanent --zone=public --add-service=rpc-bind
firewall-cmd --reload
```

NFS客户端（在K8s节点服务器上执行）：

```
mkdir -p /home/pv
echo '172.16.200.155:/home/nfsshare /home/pv nfs defaults 0 0' >> /etc/fstab
mount -a
```

使用 ansible 脚本来创建动态 pv （在 172.16.200.151 上执行）：

```
# 生成自定义配置文件
ansible-playbook /etc/ansible/tools/init_vars.yml

# 根据实际信息修改
vi /etc/ansible/roles/cluster-storage/vars/main.yml

# 创建 nfs provisioner
ansible-playbook /etc/ansible/roles/cluster-storage/cluster-storage.yml

# 执行成功后验证
kubectl get pod --all-namespaces | grep nfs-prov
```

Mysql

创建本地持久存储卷 `local_volume1.yaml` ，用来存储Mysql数据库文件（这里没有用上面的NFS，因为测试发现无法启动Mysql容器）：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-volume
spec:
  capacity:
    storage: 20Gi
  accessModes:
```

```
    - ReadWriteOnce
  volumeMode: Filesystem
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: "mysql-storage-class"
  hostPath:
    path: "/home/data"
```

因为指定的是K8s节点上的路径，所以需要在节点上创建对应路径：

```
mkdir -p /home/data
```

Mysql持久存储声明 `mysql_pv_claim.yaml` ，K8s会根据 storageClassName 找到我们定义的 pv：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  storageClassName: mysql-storage-class
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

Mysql服务 `mysql_svc.yaml` ，注意我们设定了 clusterIP，它会被用到后面TARS容器的创建中：

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  type: NodePort
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 33306
      protocol: TCP
  clusterIP: 10.68.0.10
  selector:
```

```
    app: mysql
    tier: mysql
```

Mysql root密码设置：

```
kubectl create secret generic mysql-pass --from-literal=password=Passw0rd@
```

Mysql容器 `mysql_deploy.yaml` ：

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  selector:
    matchLabels:
      app: mysql
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
        tier: mysql
    spec:
      containers:
      - image: mysql:8
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
```

```
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

TARS Master

TARS主节点镜像可以根据需要选择 `tarscloud/tars` 的不同tag。

持久存储声明

`tars_pv_claim.yaml` ，K8s会根据 storageClassName 找到持久存储卷：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: tarsmaster-pv-claim
  labels:
    app: tarsmaster
spec:
  storageClassName: nfs-dynamic-class
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
```

TARS服务

`tars_svc.yaml` ，这里也指定了clusterIP，请根据需要添加更多端口：

```
apiVersion: v1
kind: Service
metadata:
  name: tarsmaster
  labels:
    app: tarsmaster
spec:
  type: NodePort
  ports:
    - port: 3000
      targetPort: 3000
      nodePort: 33000
```

```
      protocol: TCP
      name: tarsweb
    - port: 10001
      targetPort: 10001
      protocol: TCP
      name: tarsconfig
    - port: 10002
      targetPort: 10002
      protocol: TCP
      name: tarsnotify
    - port: 10003
      targetPort: 10003
      protocol: TCP
      name: tarsstat
    - port: 10004
      targetPort: 10004
      protocol: TCP
      name: tarsproperty
    - port: 10005
      targetPort: 10005
      protocol: TCP
      name: tarslog
    - port: 10006
      targetPort: 10006
      protocol: TCP
      name: tarsquerystat
    - port: 10007
      targetPort: 10007
      protocol: TCP
      name: tarsqueryproper
    - port: 17890
      targetPort: 17890
      protocol: TCP
      name: tarsregistry
    - port: 17891
      targetPort: 17891
      protocol: TCP
      name: tarsregistry2
    - port: 19385
      targetPort: 19385
      protocol: TCP
      name: tarsnode
    - port: 19386
      targetPort: 19386
      protocol: TCP
      name: tarsnode2
    - port: 20002
      targetPort: 20002
```

```
      nodePort: 33222
      protocol: TCP
      name: testsvc
  clusterIP: 10.68.0.20
  selector:
    app: tarsmaster
    tier: tarsmaster
```

TARS容器

`tars_deploy.yaml` ，注意 `DBIP` 的值使用了上面的 Mysql 容器名称 `mysql` ：

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: tarsmaster
  labels:
    app: tarsmaster
spec:
  selector:
    matchLabels:
      app: tarsmaster
      tier: tarsmaster
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: tarsmaster
        tier: tarsmaster
    spec:
      containers:
      - image: tarscloud/tars:latest
        name: tarsmaster
        env:
          - name: DBPassword
            value: "Passw0rd@"
          - name: DBIP
            value: mysql
          - name: DBUser
            value: "root"
          - name: DBPort
            value: "3306"
          - name: MOUNT_DATA
            value: "true"
        ports:
```

```
        - containerPort: 3000
          name: tarsweb
        - containerPort: 10001
          name: tarsconfig
        - containerPort: 10002
          name: tarsnotify
        - containerPort: 10003
          name: tarsstat
        - containerPort: 10004
          name: tarsproperty
        - containerPort: 10005
          name: tarslog
        - containerPort: 10006
          name: tarsquerystat
        - containerPort: 10007
          name: tarsqueryproper
        - containerPort: 17890
          name: tarsregistry
        - containerPort: 17891
          name: tarsregistry2
        - containerPort: 19385
          name: tarsnode
        - containerPort: 19386
          name: tarsnode2
        - containerPort: 20002
          name: testsvc
      volumeMounts:
        - name: tarsmaster-persistent-storage
          mountPath: /data
    volumes:
      - name: tarsmaster-persistent-storage
        persistentVolumeClaim:
          claimName: tarsmaster-pv-claim
```

部署TARS

成功后，我们可以通过 `kubectl get pods` 找到我们部署的 Pod，然后运行 `kubectl describe pod tarsmaster` 来查看TARS主节点的相关信息：

```
Name:           tarsmaster-6dd95df944-5jvg8
Namespace:      default
Node:           172.16.200.153/172.16.200.153
Start Time:     Wed, 07 Nov 2018 14:06:17 +0800
Labels:         app=tarsmaster
                pod-template-hash=6dd95df944
```

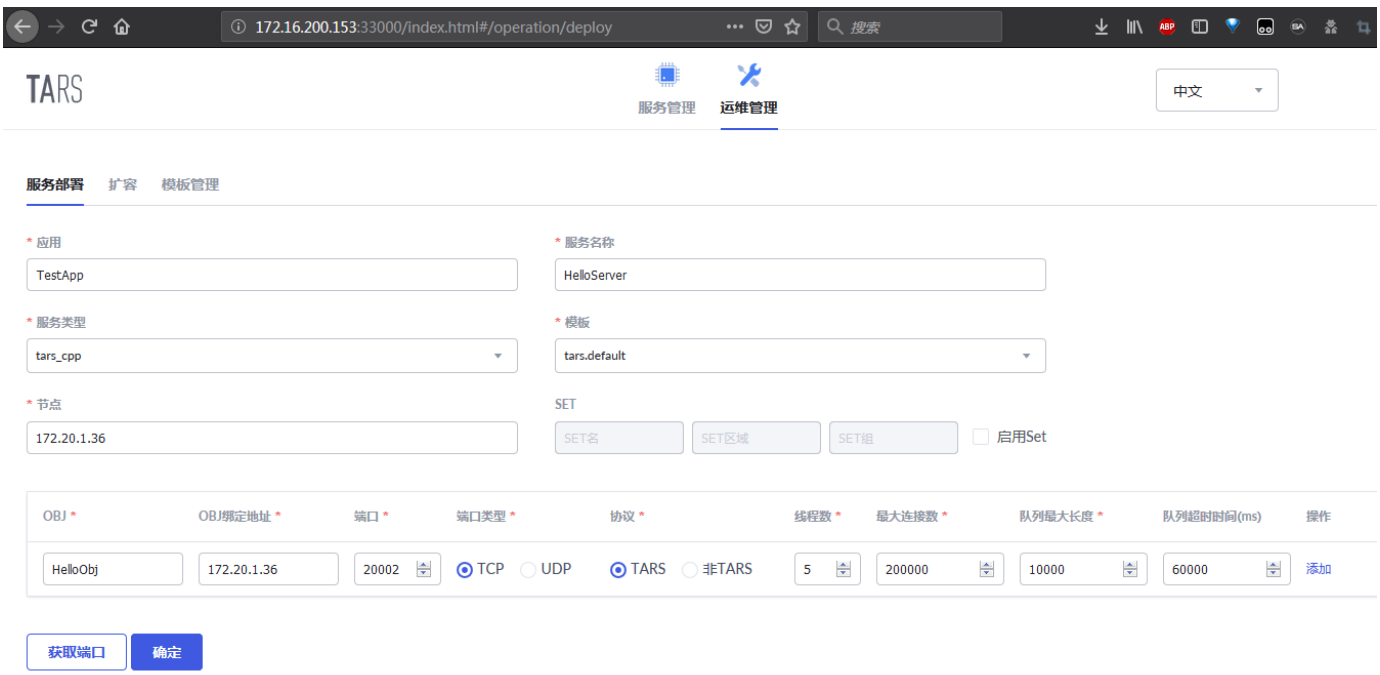
```
                tier=tarsmaster
Annotations:    <none>
Status:         Running
IP:             172.20.2.33
Controlled By:  ReplicaSet/tarsmaster-6dd95df944
Containers:
  tarsmaster:
    Container ID:
docker://9a0063f0a618cbd24432c34cc2266644a643eec71a5f6d2ff653ec6af478927c
    Image:          tarscloud/tars:latest
    Image ID:       docker-pullable://tarscloud/
tars@sha256:ae7b690a6919300e9ea0925e6fc0c587db6214fd95f665d945f8b6f1d1782119
    Ports:          3000/TCP, 10001/TCP, 10002/TCP, 10003/TCP, 10004/TCP,
10005/TCP, 10006/TCP, 10007/TCP, 17890/TCP, 17891/TCP, 19385/TCP, 19386/TCP,
20002/TCP
    Host Ports:     0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/
TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP
    State:          Running
      Started:      Wed, 07 Nov 2018 14:07:02 +0800
    Ready:          True
    Restart Count:  0
    Environment:
      DBPassword:  Passw0rd@
      DBIP:        10.68.0.10
      DBUser:      root
      DBPort:      3306
      MOUNT_DATA:  true
    Mounts:
      /data from tarsmaster-persistent-storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-tfg9p
(ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  tarsmaster-persistent-storage:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName:  tarsmaster-pv-claim
    ReadOnly:   false
  default-token-tfg9p:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-tfg9p
    Optional:    false
QoS Class:       BestEffort
```

```
Node-Selectors:  <none>
Tolerations:     <none>
Events:
  Type    Reason     Age   From                        Message
  ----    ------     ----  ----                        -------
  Normal  Scheduled  35m   default-scheduler           Successfully assigned
default/tarsmaster-68985c84d5-4p7w5 to 172.16.200.153
  Normal  Pulling    35m   kubelet, 172.16.200.153  pulling image "tarscloud/
tars:latest"
  Normal  Pulled     35m   kubelet, 172.16.200.153  Successfully pulled image
"tarscloud/tars:latest"
  Normal  Created    34m   kubelet, 172.16.200.153  Created container
  Normal  Started    34m   kubelet, 172.16.200.153  Started container
```

我们可以看到它被部署到了节点 172.16.200.153 上，我们可以通过 nodeIP:nodePort 来访问TARS的Web管理系统：http://172.16.200.153:33000 ，当然后面也可以架一个 Ingress 来作为入口。

部署应用

我们可以像在普通主机上一样通过Web管理系统部署应用到TARS，因为TARS自身实现的原因，节点IP需要使用容器的IP（这里是 172.20.1.36）。这里我们部署了一个最简单的例子应用 `TestApp.HelloServer` 。



TARS Node

TARS节点镜像可以根据需要选择 `tarscloud/tars-node` 的不同tag。

持久存储声明

`tarsnode_pv_claim.yaml` ，K8s会根据 storageClassName 找到持久存储卷：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: tarsnode-pv-claim
  labels:
    app: tarsnode
spec:
  storageClassName: nfs-dynamic-class
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
```

TARS节点服务

`tarsnode_svc.yaml` ，这里也指定了clusterIP，请根据需要添加更多端口：

```
apiVersion: v1
kind: Service
metadata:
  name: tarsnode
  labels:
    app: tarsnode
spec:
  type: NodePort
  ports:
    - port: 19385
      targetPort: 19385
      nodePort: 39385
      protocol: TCP
      name: tarsnode
    - port: 19386
      targetPort: 19386
      nodePort: 39386
      protocol: TCP
      name: tarsnode2
    - port: 20002
      targetPort: 20002
      nodePort: 33224
      protocol: TCP
      name: testsvc
  clusterIP: 10.68.0.22
  selector:
    app: tarsnode
    tier: tarsnode
```

TARS节点容器

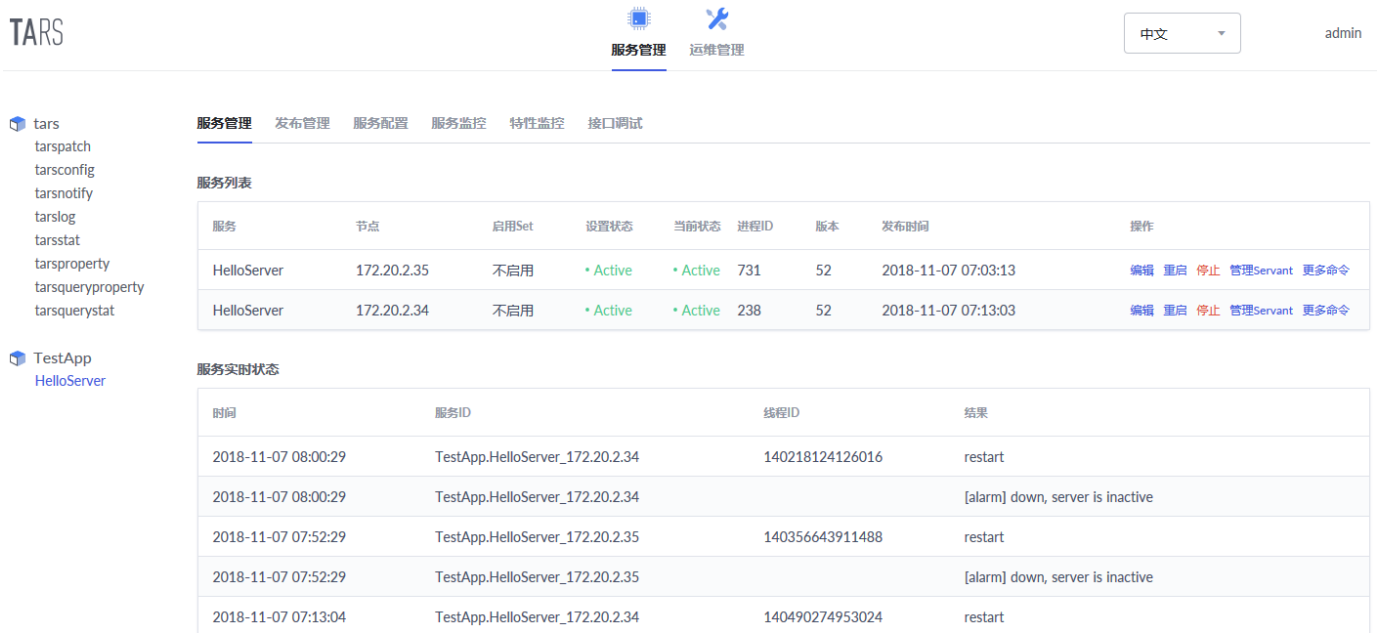
`tarsnode_deploy.yaml` ，注意我们为 `MASTER` 环境变量赋值为主节点容器名称 `tarsmaster` ：

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: tarsnode
  labels:
    app: tarsnode
spec:
  selector:
    matchLabels:
      app: tarsnode
      tier: tarsnode
```

```
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: tarsnode
        tier: tarsnode
    spec:
      containers:
      - image: tarscloud/tars-node:latest
        name: tarsnode
        env:
          - name: MASTER
            value: tarsmaster
          - name: MOUNT_DATA
            value: "true"
        ports:
          - containerPort: 19385
            name: tarsnode
          - containerPort: 19386
            name: tarsnode2
          - containerPort: 20002
            name: testsvc
        volumeMounts:
          - name: tarsnode-persistent-storage
            mountPath: /data
      volumes:
        - name: tarsnode-persistent-storage
          persistentVolumeClaim:
            claimName: tarsnode-pv-claim
```

应用扩容

部署完节点后，在主节点进行扩容，然后将应用部署到新的节点（请忽略图中的IP变化，因为做过多轮测试），这里就不做具体过程描述了：



容器的销毁与重建

我们在上面的Master容器创建YAML里设置了 `MOUNT_DATA` 为 true ，那么这个容器应该是“无状态”的。 **无状态** 打引号的原因是因为TARS本身对IP的依赖比较重，所以在K8s里销毁然后重新部署一个TARS Master，其容器IP变化后，是否还能保持原来的可用性呢？

这里需要考虑两方面的因素：1、Master本身的服务是否能正常运行在新的容器IP上；2、已经连接的Node是否能够继续与Master保持正常通信。

第一个问题，已经通过修改容器启动初始化脚本基本实现了。所以请使用最新的镜像进行K8s部署。

第二个问题，则需要用户进入Node容器（可以通过docker或者通过K8s的控制面板），杀死已有的TARS进程，然后重启 tarsnode： `cd /usr/local/app/tars && ./tarsnode_install.sh` 。

关于 TARS + K8s 的一些思考

2018-10-31:

TARS本身是一个微服务的实现与治理套件，它与K8s有一些功能重叠的地方，例如服务的部署、扩容。把TARS部署到K8s上的意义究竟有多大？这个可能见仁见智。我的想法是通过K8s让用户忽

略服务器资源的概念，通过比较简单的步骤能够让一个TARS集群跑起来，集中注意力在应用开发与部署上，而不是花太多精力在安装服务器上。

TARS的管理是很重的依赖IP的，这在运维的角度来看无可厚非。但是K8s不是这样的，它的设计更注重服务而把IP隐藏到系统治理之下。举个简单的例子，TARS里一个服务部署到某个节点，那么这个节点IP是被会记录到数据库的好几个表里的，而如果在K8s下，容器的IP是自动分配的，如果扩容或者删除重建容器，容器IP一般都会变化，这个时候TARS的主控肯定是找不到节点的。

一种可能的解决方案是使用容器名Hostname来代替IP的作用，从TARS原始的配置文件来看是支持hostname方式的。但是目前的框架实现里，有很多地方都是通过获取IP来修改配置文件以及通过读取数据库中的IP配置来寻找目标。这方面要做改动，工作量比较大。

另一种方案是利用K8s比较新的版本里对Pod固定IP的支持，然后想办法固定Pod里容器的IP，或者让容器能够使用Pod的固定IP。固定容器IP的想法，唯品会在他们的《唯品会Noah云平台实现内幕披露》里介绍了”容器本地rebuild & 容器固定IP”的进展，但看起来还是固定Pod的IP；他们之前的唯品会基于Kubernetes（k8s）网络方案演进一文最后提到了对kubelet也进行了改造，主要包括根据Pod Spec中指定IP进行相关的容器创建（docker run加入IP指定）以及Pod删除时释放IP操作，这个方案看起来是比较可行的。但是对kubelet改造的工作量也不小，而且K8s进展这么快，Hack源码不是一个好的办法。

综上，比较合理的方案是在TARS框架里加入对Hostname的支持。在现阶段还没有这方面支持的情况下，可以考虑在容器的初始化脚本里加入对数据库里IP修改的的功能。

2018-11-1:

与明杰在微信里进行了简单的讨论，他们的方案设想是使用K8s插件来监听资源变化，利用K8s的Watch机制来实现Pod变化时，把变化同步到TARS管理平台。这里的Pod变化应该就是IP变化。

我还没有仔细研究K8s的Watch机制，粗略的构思应该是基于某个开源网络插件，定制一个为TARS的网络插件，当Pod被销毁或创建时，把对应的TARS服务IP变化更新到管理平台上去。这个方案跟上面的第二个方案差不多，不同的是它不固定IP，而是修改TARS管理平台上的记录，这一点跟我最后的想法又是一样的。

目前的进展是，TARS容器的初始化脚本可以置入IP修改SQL语句，这样容器被重新创建时，会判断挂载的 /data 目录下是否有旧IP记录，如果有就调用SQL来修改数据库记录。这块的测试还没有完成，但是从原理判断问题不大。扩容的节点，如果销毁重建后IP变化了，可以认为是新节点，将老节点信息删掉，加入新节点信息到TARS管理平台即可。

目前碰到的问题是用TARS的Web管理平台扩容出了问题……

2018-11-7:

Web管理平台的bug已经被官方修复了，找时间把后面的步骤做完。

我们在上面的Master容器创建YAML里设置了 `MOUNT_DATA` 为 true ，那么这个容器应该是“无状态”的。 **无状态** 打引号的原因是因为TARS本身对IP的依赖比较重，所以在K8s里销毁然后重新部署一个TARS Master，其容器IP变化后，是否还能保持原来的可用性呢？

这里需要考虑两方面的因素：1、Master本身的服务是否能正常运行在新的容器IP上；2、已经连接的Node是否能够继续与Master保持正常通信。

第一个问题，已经通过修改容器启动初始化脚本基本实现了；第二个问题，则需要用户进入Node容器（可以通过docker或者通过K8s的控制面板），杀死已有的TARS进程，然后重启 tarsnode：`cd /usr/local/app/tars && ./tarsnode_install.sh` 。