Helm&Dashboard&prometheus

一、Helm

Helm 是 Kubernetes 的软件包管理工具。本文需要读者对 Docker、Kubernetes 等相关知识有一定的了解。 本文将介绍 Helm 中的相关概念和基本工作原理，并通过一些简单的示例来演示如何使用Helm来安装、升级、回滚一个 Kubernetes 应用。

1、helm的理解

1.1、Helm 是什么?

Helm 是 Kubernetes 的包管理器。包管理器类似于我们在 Ubuntu 中使用的apt、Centos中使用的 yum 或者Python中的 pip 一样，能快速查找、下载和安装软件包。Helm 由客户端组件 helm 和服务端组件 Tiller 组成, 能够将一组K8S资源打包统一管理, 是查找、共享和使用为Kubernetes构建的软件的最佳方式。

1.2、Helm 解决了什么痛点?

在 Kubernetes中部署一个可以使用的应用，需要涉及到很多的 Kubernetes 资源的共同协作。比如你安装一个 WordPress 博客，用到了一些 Kubernetes (下面全部简称k8s)的一些资源对象，包括 Deployment 用于部署应用、Service 提供服务发现、Secret 配置 WordPress 的用户名和密码，可能还需要 pv 和 pvc 来提供持久化服务。并且 WordPress 数据是存储在mariadb里面的，所以需要 mariadb 启动就绪后才能启动 WordPress。这些 k8s 资源过于分散，不方便进行管理，直接通过 kubectl 来管理一个应用，你会发现这十分蛋疼。
所以总结以上，我们在 k8s 中部署一个应用，通常面临以下几个问题：

- 如何统一管理、配置和更新这些分散的 k8s 的应用资源文件
- 如何分发和复用一套应用模板
- 如何将应用的一系列资源当做一个软件包管理

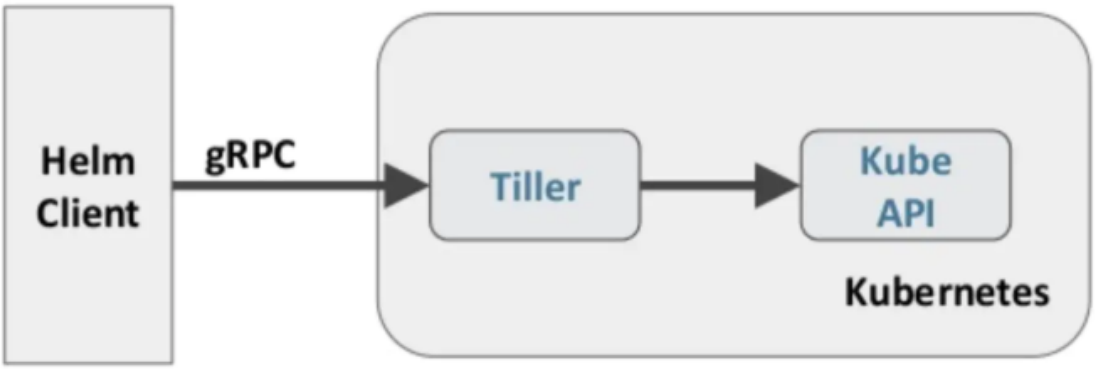
1.3、Helm 相关组件及概念

Helm 包含两个组件，分别是 helm 客户端 和 Tiller 服务器：

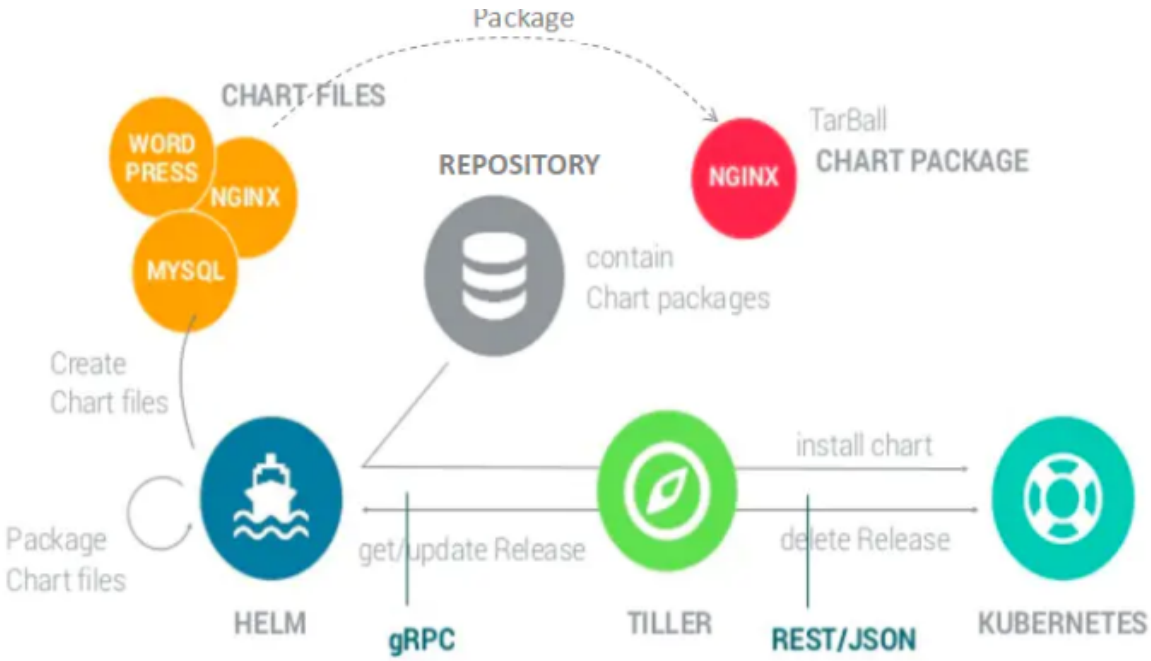
- **helm** 是一个命令行工具，用于本地开发及管理chart，chart仓库管理等
- **Tiller** 是 Helm 的服务端。Tiller 负责接收 Helm 的请求，与 k8s 的 apiserver 交互，根据chart 来生成一个 release 并管理 release
- **chart** Helm的打包格式叫做chart，所谓chart就是一系列文件, 它描述了一组相关的 k8s 集群资源
- **release** 使用 helm install 命令在 Kubernetes 集群中部署的 Chart 称为 Release
- Repoistory Helm chart 的仓库，Helm 客户端通过 HTTP 协议来访问存储库中 chart 的索引文件和压缩包

1.4、Helm 原理

下面两张图描述了 Helm 的几个关键组件 Helm（客户端）、Tiller（服务器）、Repository（Chart 软件仓库）、Chart（软件包）之间的关系以及它们之间如何通信,如下图（helm通信组件）



1.5、helm架构



创建release

- helm 客户端从指定的目录或本地tar文件或远程repo仓库解析出chart的结构信息
- helm 客户端指定的 chart 结构和 values 信息通过 gRPC 传递给 Tiller
- Tiller 服务端根据 chart 和 values 生成一个 release
- Tiller 将install release请求直接传递给 kube-apiserver

删除release

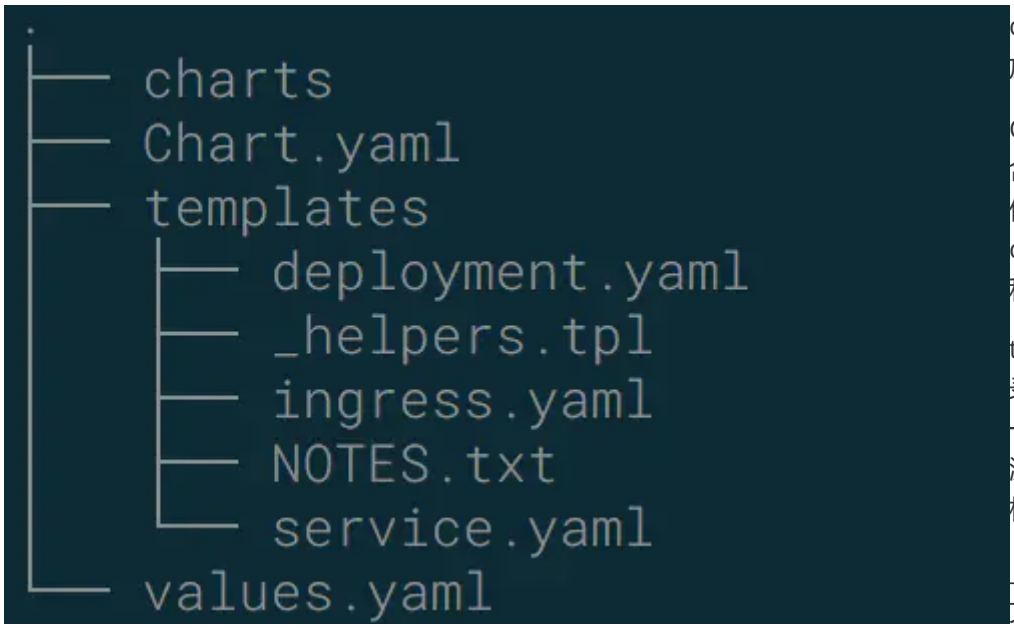
- helm 客户端从指定的目录或本地tar文件或远程repo仓库解析出chart的结构信息
- helm 客户端指定的 chart 结构和 values 信息通过 gRPC 传递给 Tiller
- Tiller 服务端根据 chart 和 values 生成一个 release
- Tiller 将delete release请求直接传递给 kube-apiserver

更新release

- helm 客户端将需要更新的 chart 的 release 名称 chart 结构和 value 信息传给 Tiller
- Tiller 将收到的信息生成新的 release，并同时更新这个 release 的 history
- Tiller 将新的 release 传递给 kube-apiserver 进行更新

1.6、chart 的基本结构

Helm的打包格式叫做chart，所谓chart就是一系列文件, 它描述了一组相关的 k8s 集群资源。Chart中的文件安装特定的目录结构组织, 最简单的chart 目录如下所示：



charts 目录存放依赖的chart

Chart.yaml 包含Chart的基本信息，包括chart版本，名称等

templates 目录下存放应用一系列 k8s 资源的 yaml 模板

_helpers.tpl 此文件中定义一些可重用的模板片断，此文件中的定义在任何资源定义模板中可用

NOTES.txt 介绍chart 部署后的帮助信息，如何使用chart等

values.yaml 包含了必要的值定义（默认值）, 用于存储 templates 目录中模板文件中用到变量的值

2、helm安装

Helm 提供了几种安装方式，本文提供两种安装方式，想要查看更多安装方式，请阅读 Helm 的官方文档:

```
# ======= 第一种安装方式：  手动方式安装 ========
# 下载 Helm 二进制文件
wget https://storage.googleapis.com/kubernetes-helm/helm-v2.9.1-linux-
amd64.tar.gz

# 解压缩
tar -zxvf helm-v2.9.1-linux-amd64.tar.gz

# 复制 helm 二进制 到bin目录下
cp linux-amd64/helm /usr/local/bin/

# ======= 第二种安装方式：  使用官方二进制脚本进行安装 ========
curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get >
get_helm.sh
chmod 700 get_helm.sh
./get_helm.sh
# 你还可以通过 Helm 的 github 项目下找到你想要的 Helm 版本的二进制，然后通过手动安装方式一样安装即可
```

3、tiller安装

安装好 helm 客户端后，就可以通过以下命令将 Tiller 安装在 kubernetes 集群中：

```
# 它由客户端helm和服务端tiller组成，而helm3.0之后它去掉了tiller，而直接与k8s通讯，可以说在部署上更简单了，而今天我们主要还是部署2.x版本的helm.
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system


# 构建rbac
kubectl create -f rbac-config.yaml

# 初始化tiller
helm init --service-account tiller --skip-refresh

# 查看tiller的pod信息
kubectl get pods -n kube-system | grep tiller
# 到现在为止，我们的helm就安装功能了，之后我们装运行helm来进行charts的安装
# Pulling image "gcr.io/kubernetes-helm/tiller:v2.15.2 发现下载镜像tiller:v2.15失败

# 镜像从阿里云平台下载，然后命名为使用的默认的镜像：gcr.io/kubernetes-helm/tiller:v2.15.2
docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/tiller:v2.15.2
# 打包镜像
docker tag registry.cn-hangzhou.aliyuncs.com/google_containers/tiller:v2.15.2 gcr.io/kubernetes-helm/tiller:v2.15.2
# 然后把镜像上传到其他2个node节点即可，tiller安装将会优先使用本地镜像

# 查询helm是否安装成功
helm version
Client: &version.Version{SemVer:"v2.15.2", GitCommit:"8dce272473e5f2a7bf58ce79bb5c3691db54c96b", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.15.2", GitCommit:"8dce272473e5f2a7bf58ce79bb5c3691db54c96b", GitTreeState:"clean"}


# 其他的安装方式：  可以指定阿里云镜像
helm init --upgrade -i registry.cn-hangzhou.aliyuncs.com/google_containers/tiller:v2.15.2 --stable-repo-url https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts
```

4、helm自定义安装

```
# 创建文件夹
mkdir hello-helm
cd hello-helm
# 创建自描述文件 Chart.yaml , 这个文件必须有 name 和 version 定义
cat << 'EOF' > ./Chart.yaml
name: hello-world
version: 1.0.0
EOF

# 创建模板文件， 用于生成 Kubernetes 资源清单（manifests）
mkdir ./templates
cat <<'EOF' > ./templates/deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-world
        image: hub.kaikeba.com/library/myapp:v1
        ports:
        - containerPort: 80
          protocol: TCP
EOF

cat <<'EOF' > ./templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: hello-world
EOF

# 使用命令 helm install RELATIVE_PATH_TO_CHART 创建一次Release
# 安装完毕即可在外网进行访问
helm install .

# 列出已经部署的 Release
helm ls
# 查询一个特定的 Release 的状态
```

```
helm status RELEASE_NAME
# 移除所有与这个 Release 相关的 Kubernetes 资源
helm delete RELEASE_NAME
# helm rollback RELEASE_NAME REVISION_NUMBER
helm rollback RELEASE_NAME 1
# 使用 helm delete --purge RELEASE_NAME 移除所有与指定 Release 相关的 Kubernetes 资源和所有这个 Release 的记录
helm delete --purge RELEASE_NAME
helm ls --deleted

# 修改配置文件更新
helm upgrade RELEASE_NAME .

# 配置体现在配置文件 values.yaml
cat <<'EOF' > ./values.yaml
image:
  repository: hub.kaikeba.com/java12/myapp
  tag: 'v1'
EOF
# 这个文件中定义的值，在模板文件中可以通过 .VAlues对象访问到
cat <<'EOF' > ./templates/deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-world
        image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
        ports:
        - containerPort: 8080
          rotocol: TCP
EOF
# 在 values.yaml 中的值可以被部署 release 时用到的参数 --values YAML_FILE_PATH 或 --set key1=value1, key2=value2 覆盖掉
helm install --set image.tag='v3' .
helm upgrade RELEASE_NAME --set image.tag='v3' .
# 升级版本
helm upgrade   -f values.yaml test .
```

5、Debug

```
# 使用模板动态生成K8s资源清单，非常需要能提前预览生成的结果。
# 使用--dry-run --debug 选项来打印出生成的清单文件内容，而不执行部署
helm install . --dry-run --debug --set image.tag=latest
```

二、Dashboard

```
# 将远程chaart下载到本地
# 更新一下helm下载的仓库地址源
helm repo update
# 使用fetch开始下载
helm fetch  stable/kubernetes-dashboard
# 下载包如下所示，解压即可
kubernetes-dashboard-1.10.1.tgz

# 编辑dashboard.yaml文件
image:
  repository: registry.cn-hangzhou.aliyuncs.com/google_containers/kubernetes-
dashboard-amd64
  tag: v1.10.0
ingress:
  enabled: true
  hosts:
    - k8s.frognew.com
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
  tls:
    - secretName: frognew-com-tls-secret
      hosts:
      - k8s.frognew.com
rbac:
  clusterAdminRole: true

# 开始安装，注意：  在安装的时候需要使用dashboard相关镜像，因此解决科学上网的问题，这里可以上传本地准备好的dashbord的镜像，上传到node节点即可。
helm install . \
 -n kubernetes-dashboard \
 --namespace kube-system \
 -f dashboard.yaml

# 将svc的ClusterIIP改为NotePort格式进行访问
#提供外界访问，修改成nodePort
kubectl get svc -n kube-system
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
         AGE
kube-dns               ClusterIP   10.96.0.10      <none>
53/UDP,53/TCP,9153/TCP   4d2h
kubernetes-dashboard   ClusterIP   10.110.55.113   <none>        443/TCP
         5m10s
tiller-deploy          ClusterIP   10.97.87.192    <none>        44134/TCP
         114m

kubectl edit svc kubernetes-dashboard -n kube-system
service/kubernetes-dashboard edited    #type: NodePort

kubectl get svc -n kube-system
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
         AGE
```

```
kube-dns               ClusterIP   10.96.0.10      <none>
53/UDP,53/TCP,9153/TCP   4d2h
kubernetes-dashboard   NodePort    10.110.55.113   <none>        443:31147/TCP
         6m54s
tiller-deploy          ClusterIP   10.97.87.192    <none>        44134/TCP

#浏览器访问
<a href="https://10.0.0.11:31147">https://10.0.0.11:31147</a>

# 查看令牌名称
kubectl -n kube-system get secrets |grep kubernetes-dashboard-token
# 查看令牌
kubectl describe secrets kubernetes-dashboard-token-czmt9  -n kube-system
```

三、prometheus

Prometheus github 地址：https://github.com/coreos/kube-prometheus

1、组件说明

1）MetricServer：是kubernetes集群资源使用情况的聚合器，收集数据给kubernetes集群内使用，如kubectl,hpa,scheduler等。

2）PrometheusOperator：是一个系统监测和警报工具箱，用来存储监控数据。

3）NodeExporter：用于各node的关键度量指标状态数据。

4）KubeStateMetrics：收集kubernetes集群内资源对象数据，制定告警规则。

5）Prometheus：采用pull方式收集apiserver，scheduler，controller-manager，kubelet组件数据，通过http协议传输。

6）Grafana：是可视化数据统计和监控平台

2、构建记录

```
# 1）下载相关配置文件
git clone https://github.com/coreos/kube-prometheus.git
cd kube-prometheus/manifests/

# 2) 修改 grafana-service.yaml 文件，使用 nodepode 方式访问 grafana
# vim grafana-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: grafana
  name: grafana
  namespace: monitoring
spec:
  type: NodePort      #添加内容
```

```
  ports:
  - name: http
    port: 3000
    targetPort: http
    nodePort: 30100   #添加内容
  selector:
    app: grafana

# 3）修改 prometheus-service.yaml，改为 nodepode
# vim prometheus-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    prometheus: k8s
  name: prometheus-k8s
  namespace: monitoring
spec:
  type: NodePort  #添加
  ports:
  - name: web
    port: 9090
    targetPort: web
    nodePort: 30200  #添加
  selector:
    app: prometheus
    prometheus: k8s
  sessionAffinity: ClientIP

# 4)修改 alertmanager-service.yaml，改为 nodepode
# vim alertmanager-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    alertmanager: main
  name: alertmanager-main
  namespace: monitoring
spec:
  type: NodePort  #添加
  ports:
  - name: web
    port: 9093
    targetPort: web
    nodePort: 30300  #添加
  selector:
    alertmanager: main
    app: alertmanager
  sessionAffinity: ClientIP

  # 开始执行安装 注意：执行之前必须把promutheus相关镜像导入本地镜像仓库（三个节点都必须导入），当然也可以从网络下载，但是必须解决科学上网的问题。
  cd /root/k8s/prometheus/kube-prometheus/manifests
  kubectl apply -f .  #多运行几遍,以便于查看是否安装成功

# 找不到命名空间，解决方法如下
#1、直接创建monitoring命令空间
 kubectl create namespace monitoring
```

```
#2、通过yml文件创建monitoring命令空间
apiVersion: v1
kind: Namespace
metadata:
   name: monitoring
   labels:
     name: monitoring

  # 查看是否安装成功
  kubectl get pod -n monitoring

  # 测试监控node节点
   kubectl top node

  # 测试监控pod
  kubectl top pod

# prometheus访问测试
# prometheus 对应的 nodeport 端口为 30200，访问http://MasterIP:30200
# 通过访问http://MasterIP:30200/target可以看到 prometheus 已经成功连接上了 k8s 的
apiserver


# grafana访问测试,添加数据来源：默认已经添加好
http://10.0.0.11:30100/login    admin    admin   （默认要修改密码）
```