01 不妨聊聊各个组件

> `Components官网`：https://istio.io/docs/ops/deployment/architecture/#components

1.1 Proxy[Envoy]

> **Proxy在Istio架构中必须要有**
>
> Envoy是由Lyft开发并开源，使用C++编写的高性能代理，负责在服务网格中服务的进出流量。
>
> ```
> Istio uses an extended version of the Envoy proxy. Envoy is a high-performance proxy developed in C++ to mediate all inbound and outbound traffic for all services in the service mesh. Envoy proxies are the only Istio components that interact with data plane traffic.
> ```
>
> `官网`：https://www.envoyproxy.io/
>
> ```
> ENVOY IS AN OPEN SOURCE EDGE AND SERVICE PROXY, DESIGNED FOR CLOUD-NATIVE APPLICATIONS
> ```
>
> `github`：https://github.com/envoyproxy/envoy
>
> ```
> Envoy is hosted by the Cloud Native Computing Foundation (CNCF). If you are a company that wants to help shape the evolution of technologies that are container-packaged, dynamically-scheduled and microservices-oriented, consider joining the CNCF. For details about who's involved and how Envoy plays a role, read the CNCF announcement.
> ```

1.1.1 Features

- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics

1.1.2 为什么选择Envoy?

对于Sidecar/Proxy其实不仅仅可以选择Envoy，还可以用Linkerd、Nginx和NginMesh等。

像Nginx作为分布式架构中比较广泛使用的网关，Istio默认却没有选择，是因为Nginx没有Envoy优秀的配置扩展，Envoy可以实时配置。

1.2 Mixer

> **Mixer在Istio架构中不是必须的**

官网：https://istio.io/docs/ops/deployment/architecture/#mixer

```
Mixer is a platform-independent component. Mixer enforces access control
and usage policies across the service mesh, and collects telemetry data
from the Envoy proxy and other services. The proxy extracts request level
attributes, and sends them to Mixer for evaluation. You can find more
information on this attribute extraction and policy evaluation in our Mixer
Configuration documentation.

Mixer includes a flexible plugin model. This model enables Istio to
interface with a variety of host environments and infrastructure backends.
Thus, Istio abstracts the Envoy proxy and Istio-managed services from these
details.
```

- 为集群执行访问控制，哪些用户可以访问哪些服务，包括白名单检查、ACL检查等
- 策略管理，比如某个服务最多只能接收多少流量请求
- 遥测报告上报，比如从Envoy中收集数据[请求数据、使用时间、使用的协议等]，通过Adpater上报给Promethues、Heapster等

1.3 Pilot

Pilot在Istio架构中必须要有

官网：https://istio.io/docs/ops/deployment/architecture/#pilot

```
Pilot provides service discovery for the Envoy sidecars, traffic management
capabilities for intelligent routing (e.g., A/B tests, canary rollouts,
etc.), and resiliency (timeouts, retries, circuit breakers, etc.).
```

Pilot为Envoy sidecar提供了服务发现功能，为智能路由提供了流量管理能力(比如A/B测试、金丝雀发布等)。

Pilot本身不做服务注册，它会提供一个接口，对接已有的服务注册系统，比如Eureka，Etcd等。

Pilot对配置的格式做了抽象，整理成能够符合Envoy数据层的API。

```
(1)Polit定了一个抽象模型，从特定平台细节中解耦，用于对接外部的不同平台
(2)Envoy API负责和Envoy的通讯，主要是发送服务发现信息和流量控制规则给Envoy
(3)Platform Adapter是Pilot抽象模型的实现版本，用于对接外部的不同平台
...
```

1.4 Galley

Galley在Istio架构中不是必须的

官网：https://istio.io/docs/ops/deployment/architecture/#galley

```
Galley is Istio's configuration validation, ingestion, processing and
distribution component. It is responsible for insulating the rest of the
Istio components from the details of obtaining user configuration from the
underlying platform (e.g. Kubernetes).
```

主要负责istio配置的校验、各种配置之间统筹，为istio提供配置管理服务。
通过kubernetes的webhook机制对pilot和mixer的配置进行验证。

1.5 Citadel

> **Citadel在Istio架构中不是必须的**
>
> 官网：https://istio.io/docs/ops/deployment/architecture/#citadel
>
> ```
> Citadel enables strong service-to-service and end-user authentication with
> built-in identity and credential management. You can use Citadel to upgrade
> unencrypted traffic in the service mesh. Using Citadel, operators can
> enforce policies based on service identity rather than on relatively
> unstable layer 3 or layer 4 network identifiers. Starting from release 0.5,
> you can use Istio's authorization feature to control who can access your
> services.
> ```
>
> 在有一些场景中，对于安全要求是非常高的，比如支付，所以Citadel就是用来保证安全的。

02 Bookinfo

> 官网：https://istio.io/docs/examples/bookinfo/
>
> ```
> This example deploys a sample application composed of four separate
> microservices used to demonstrate【/'demənstreɪt/ 证明】 various Istio
> features.
> The application displays information about a book, similar to a single
> catalog entry of an online book store. Displayed on the page is a
> description of the book, book details (ISBN, number of pages, and so on),
> and a few book reviews.
> ```
>
> The Bookinfo application is broken into four separate microservices:
>
> - `productpage`. The `productpage` microservice calls the `details` and `reviews` microservices to populate the page.
> - `details`. The `details` microservice contains book information.
> - `reviews`. The `reviews` microservice contains book reviews. It also calls the `ratings` microservice.
> - `ratings`. The `ratings` microservice contains book ranking information that accompanies a book review.
>
> There are 3 versions of the `reviews` microservice:
>
> - Version v1 doesn't call the `ratings` service.
> - Version v2 calls the `ratings` service, and displays each rating as 1 to 5 black stars.
> - Version v3 calls the `ratings` service, and displays each rating as 1 to 5 red stars.
>
> This application is polyglot, i.e., the microservices are written in different languages. It's worth noting that these services have no dependencies on Istio, but make an interesting service mesh example, particularly because of the multitude of services, languages and versions for the `reviews` service.

2.1 理解bookinfo

(1)**productpage是Python**语言编写的，用于前端页面展示，会调用reviews微服务和details微服务

(2)**details是Ruby**语言编写的，是书籍的详情信息

(3)**reviews是Java语言编写的，是书籍的评论信息，会调用ratings微服务，有3个版本**

(4)**ratings是nodejs语言编写的，是书籍的评分信息**

2.2 sidecar自动注入到微服务

> 官网：https://istio.io/docs/examples/bookinfo/#start-the-application-services
>
> To run the sample with Istio requires **no changes to the application itself**. Instead, you simply need to configure and run the services in an Istio-enabled environment, with Envoy sidecars injected along side each service. The resulting deployment will look like this:
>
> All of the microservices will be packaged with an Envoy sidecar that intercepts **incoming and outgoing calls for the services**, providing the hooks needed to externally control, via the Istio control plane, routing, telemetry collection, and policy enforcement for the application as a whole.

(1)Change directory to the root of the Istio installation.

```
cd istio-1.0.6
```

(2)The default Istio installation uses **automatic sidecar injection**. Label the namespace that will host the application with `istio-injection=enabled`:

```
kubectl label namespace default istio-injection=enabled
kubectl get namespaces --show-labels
```

(3)Deploy your application using the `kubectl` command:

若镜像拉取不下来，可以用我的，记得打tag，rmi

```
docker pull registry.cn-hangzhou.aliyuncs.com/istio-k8s/examples-bookinfo-details-v1:1.8.0
docker pull registry.cn-hangzhou.aliyuncs.com/istio-k8s/examples-bookinfo-ratings-v1:1.8.0
docker pull registry.cn-hangzhou.aliyuncs.com/istio-k8s/examples-bookinfo-reviews-v1:1.8.0
docker pull registry.cn-hangzhou.aliyuncs.com/istio-k8s/examples-bookinfo-reviews-v2:1.8.0
docker pull registry.cn-hangzhou.aliyuncs.com/istio-k8s/examples-bookinfo-reviews-v3:1.8.0
docker pull registry.cn-hangzhou.aliyuncs.com/istio-k8s/examples-bookinfo-productpage-v1:1.8.0
```

```
kubectl apply -f istio-1.0.6/samples/bookinfo/platform/kube/bookinfo.yaml
```

(4)查看pod

```
kubectl get pods
```

```
NAME                                 READY   STATUS
details-v1-d458c8599-l2rpr           2/2     Running
productpage-v1-79d85ff9fc-jvlrn      2/2     Running
ratings-v1-567bfb85cf-lrkpm          2/2     Running
reviews-v1-fd6c96c74-ncl8k           2/2     Running
reviews-v2-68d98477f6-s5f7l          2/2     Running
reviews-v3-8495f5f6bb-mx8q2          2/2     Running
```

> (5)查看svc

```
kubectl get svc
```

```
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
details       ClusterIP   10.103.89.119   <none>        9080/TCP   5m48s
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP    18d
productpage   ClusterIP   10.104.220.93   <none>        9080/TCP   5m48s
ratings       ClusterIP   10.106.48.89    <none>        9080/TCP   5m48s
reviews       ClusterIP   10.98.5.206     <none>        9080/TCP   5m48s
```

> (6)测试一下是否成功

```
kubectl exec -it $(kubectl get pod -l app=ratings -o
jsonpath='{.items[0].metadata.name}') -c ratings -- curl
productpage:9080/productpage | grep -o "<title>.*</title>"
```

```
<title>Simple Bookstore App</title>
```

2.3 通过ingress方式访问

> (1)创建ingress规则
>
> 网盘/课堂源码/productpage-ingress.yaml

```
#ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: productpage-ingress
spec:
  rules:
  - host: productpage.istio.itcrazy2016.com
    http:
      paths:
      - path: /
        backend:
          serviceName: productpage
          servicePort: 9080
```

> (2)访问测试
>
> productpage.istio.itcrazy2016.com

2.4 通过istio的ingressgateway访问

> 官网：https://istio.io/docs/examples/bookinfo/#determine-the-ingress-ip-and-port

> (1)Define the ingress gateway for the application:
>
> istio-1.0.6/samples/bookinfo/networking/bookinfo-gateway.yaml
>
> 可以查看一下该yaml文件，一个是Gateway，一个是VirtualService

```
kubectl apply -f bookinfo-gateway.yaml
```

> (2)Confirm the gateway has been created:

```
kubectl get gateway
```

> (3)set the `INGRESS_HOST` and `INGRESS_PORT` variables for accessing the gateway

```
export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n istio-system -o jsonpath='{.items[0].status.hostIP}')
```

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
```

> (4)Set `GATEWAY_URL`:

```
export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

> (5)查看INGRESS_PORT端口

```
# 比如结果为31380
env | grep INGRESS_PORT
```

> (6)测试
>
> 不断访问测试，发现会访问的review的不同版本

```
http://121.41.10.13:31380/productpage
```

> (6)Apply default destination rules
>
> 官网：https://istio.io/docs/examples/bookinfo/#apply-default-destination-rules
>
> Before you can use Istio to control the Bookinfo version routing, you need to define the available versions, called *subsets*, in destination rules
>
> istio-1.0.6/samples/bookinfo/networking/destination-rule-all.yaml

```
kubectl apply -f destination-rule-all.yaml
```

2.5 体验Istio的流量管理

> **流量这块就体现了Pilot和Envoy的功能**

2.3.1 基于版本的路由

> 官网：https://istio.io/docs/tasks/traffic-management/request-routing/#apply-a-virtual-service
>
> 之前刷新productpage页面的时候，发现review的版本一直会变，能不能一直访问某个版本呢？比如v3
>
> istio-1.0.6/samples/bookinfo/networking/virtual-service-reviews-v3.yaml
>
> kubectl apply -f virtual-service-reviews-v3.yaml
>
> 再次访问测试：http://121.41.10.13:31380/productpage

2.5.2 基于用户身份的路由

> 官网：https://istio.io/docs/tasks/traffic-management/request-routing/#route-based-on-user-identity
>
> ```
> Next, you will change the route configuration so that all traffic from a specific user is routed to a specific service version. In this case, all traffic from a user named Jason will be routed to the service reviews:v2.
> ```
>
> ```
> Note that Istio doesn't have any special, built-in understanding of user identity. This example is enabled by the fact that the productpage service adds a custom end-user header to all outbound HTTP requests to the reviews service.
> ```
>
> Remember, `reviews:v2` is the version that includes the star ratings feature.

> (1)根据对应文件创建资源

```
istio-1.0.6/samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

> (2)测试

```
# 使用jason来登录[右上角有Sign in的功能或者url?u=jason]，一直会访问到v2版本
On the /productpage of the Bookinfo app, log in as user jason.
Refresh the browser. What do you see? The star ratings appear next to each review.
```

```
# 使用其他用户登录，一直会访问到v1版本
Log in as another user (pick any name you wish).
Refresh the browser. Now the stars are gone. This is because traffic is routed to reviews:v1 for all users except Jason.
```

2.5.3 基于权重的路由

> (1)根据对应文件创建资源
>
> istio-1.0.6/samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml
>
> 一半几率访问到v1，一半几率访问到v3
>
> 这里就相当于之前的蓝绿部署、AB测试或灰度发布
>
> 权重加起来必须是100

```
kubectl apply -f virtual-service-reviews-50-v3.yaml
```

(2)测试

```
http://121.41.10.13:31380/productpage
```

2.5.4 故障注入

官网：https://istio.io/docs/tasks/traffic-management/fault-injection/

Apply application version routing by either performing the request routing task or by running the following commands:

istio-1.0.6/samples/bookinfo/networking/

```
kubectl apply -f virtual-service-all-v1.yaml
kubectl apply -f virtual-service-reviews-test-v2.yaml
```

With the above configuration, this is how requests flow:

```
productpage → reviews:v2 → ratings (only for user jason)
productpage → reviews:v1 (for everyone else)
```

To test the Bookinfo application microservices for resiliency, inject a 7s delay between the `reviews:v2` and `ratings` microservices for user `jason`. This test will uncover a bug that was intentionally introduced into the Bookinfo app.

(1)创建一个故障注入规则，使得jason用户访问v2到ratings有7秒种的延迟

```
kubectl apply -f virtual-service-ratings-test-delay.yaml
```

(2)使用jason账户登录，并且访问productpage页面，会得到这样的一个返回信息

```
Error fetching product reviews!
Sorry, product reviews are currently unavailable for this book.
```

(3)View the web page response times:

```
Open the Developer Tools menu in you web browser.
Open the Network tab
Reload the /productpage web page. You will see that the page actually loads in
about 6 seconds.
```

2.5.5 流量迁移

官网：https://istio.io/docs/tasks/traffic-management/traffic-shifting/

This task shows you how to gradually migrate traffic from one version of a microservice to another. For example, you might migrate traffic from an older version to a new version.

> A common use case is to migrate traffic gradually from one version of a microservice to another. In Istio, you accomplish this goal by configuring a sequence of rules that route a percentage of traffic to one service or another. In this task, you will send 50% of traffic to `reviews:v1` and 50% to `reviews:v3`. Then, you will complete the migration by sending 100% of traffic to `reviews:v3`.

> (1)让所有的流量都到v1

```
kubectl apply -f virtual-service-all-v1.yaml
```

> (2)将v1的50%流量转移到v3

```
kubectl apply -f virtual-service-reviews-50-v3.yaml
```

> (3)确保v3版本没问题之后，可以将流量都转移到v3

```
kubectl apply -f virtual-service-reviews-v3.yaml
```

> (4)访问测试，看是否都访问的v3版本

2.6 体验Istio的Observe

> **这块就体现了Mixer和Envoy的功能**

2.6.1 收集Metrics

> 官网：https://istio.io/docs/tasks/observability/metrics/collecting-metrics/
>
> This task shows how to configure Istio to automatically gather telemetry for services in a mesh. At the end of this task, a new metric will be enabled for calls to services within your mesh.

> (1)Apply a YAML file with configuration for the new metric that Istio will generate and collect automatically.
>
> 网盘/课件源码/metrics-crd.yaml

```
kubectl apply -f metrics-crd.yaml
```

> (2)Send traffic to the sample application

```
http://121.41.10.13:31380/productpage
```

> (3)访问prometheus

```
http://prometheus.istio.itcrazy2016.com
```

> (4)根据`istio_double_request_count`进行查询

> (5)Understanding the metrics configuration
>
> https://istio.io/docs/tasks/observability/metrics/collecting-metrics/#understanding-the-metrics-configuration

2.6.2 查询Istio的metrics

> 官网：https://istio.io/docs/tasks/observability/metrics/querying-metrics/
>
> This task shows you how to query for Istio Metrics using Prometheus. As part of this task, you will use the web-based interface for querying metric values.

> (1)访问productpage

```
http://121.41.10.13:31380/productpage
```

> (2)打开prometheus界面

```
http://prometheus.istio.itcrazy2016.com
```

> (3)输入查询指标

```
istio_requests_total
```

> (4)About the Prometheus add-on

```
https://istio.io/docs/tasks/observability/metrics/querying-metrics/#about-the-prometheus-add-on
```

2.6.3 分布式追踪之Jaeger

> 官网：https://istio.io/docs/tasks/observability/distributed-tracing/overview/
>
> Distributed tracing enables users to track a request through mesh that is distributed across multiple services. This allows a deeper understanding about request latency, serialization and parallelism via visualization.

> jaeger官网：https://istio.io/docs/tasks/observability/distributed-tracing/jaeger/
>
> After completing this task, you understand how to have your application participate in tracing with Jaeger, regardless of the language, framework, or platform you use to build your application.
>
> istio-tracing-c8b67b59c-8vgrl 1/1 Running ---> 即Jaeger，默认已经安装

> (1)查看jaeger的svc

```
kubectl get svc -n istio-system | grep jae
kubectl get svc jaeger-query -n istio-system -o yaml
```

> (2)配置jaeger的ingress
>
> 网盘/课堂源码/jaeger-ingress.yaml

```
#ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: jaeger-ingress
  namespace: istio-system
```

```
spec:
  rules:
  - host: jaeger.istio.itcrazy2016.com
    http:
      paths:
      - path: /
        backend:
          serviceName: jaeger-query
          servicePort: 16686
```

> (3)浏览器访问测试

```
jaeger.istio.itcrazy2016.com
```

> (4)发送100个请求

```
for i in `seq 1 100`; do curl -s -o /dev/null
http://121.41.10.13:31380/productpage; done
```

> (5)进入到jaeger界面

选择productpage，查询详情

2.6.4 Mesh可视化之Kiali[了解一下]

> 官网：https://istio.io/docs/tasks/observability/kiali/
>
> This task shows you how to visualize different aspects of your Istio mesh.
>
> As part of this task, you install the Kiali add-on and use the web-based graphical user interface to view service graphs of the mesh and your Istio configuration objects. Lastly, you use the Kiali Public API to generate graph data in the form of consumable JSON.

03 安装Istio[Helm]

> 官网：https://istio.io/docs/setup/install/helm/

3.1 下载安装Helm

3.1.1 Helm简介

> Helm是Kubernetes的软件包管理工具，类似于Ubuntu中的apt、CentOS中的yum等。
>
> 可以快速查找、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。

3.1.2 解决的问题

比如在K8S中部署一个wordpress，需要创建deployment，service，secret、pv等。这些资源有时候不方便管理，过于分散，如果使用kubectl进行操作，发现还是比较恶心的。所以简单来说，helm就是为了解决上述问题。

3.1.3 各种名词概念

- chart

helm的打包格式叫chart，chart即一系列文件，描述了一组相关的k8s集群资源

- helm

客户端命令行工具，用于本地开发及管理chart、chart仓库等

- tiller

helm的服务端，tiller接收helm的请求，与k8s的apiserver打交道，根据chart生成一个release并且管理release

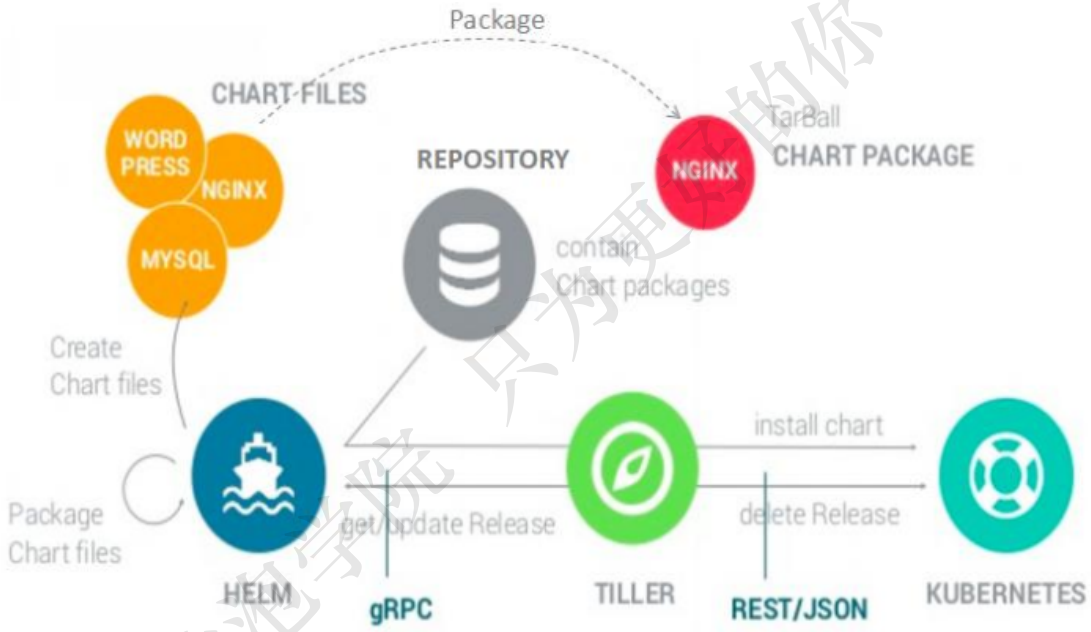
- release

helm install命令在k8s集群中部署的chart称为release

- repository helm chart

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

3.1.4 图解Helm原理



3.1.5 release操作

```
创建release
    (1)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
    (2)helm 客户端指定的 chart 结构和 values 信息通过 gRPC 传递给 Tiller
    (3)Tiller 服务端根据 chart 和 values 生成一个 release
    (4)Tiller 将install release请求直接传递给 kube-apiserver
删除release
    (1)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
    (2)helm 客户端指定的 chart 结构和 values 信息通过 gRPC 传递给 Tiller
    (3)Tiller 服务端根据 chart 和 values 生成一个 release
    (4)Tiller 将delete release请求直接传递给 kube-apiserver
更新release
    (1)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
    (2)Tiller 将收到的信息生成新的 release，并同时更新这个 release 的 history
    (3)Tiller 将新的 release 传递给 kube-apiserver 进行更新
```

3.1.6 安装Helm

> 官网：https://github.com/helm/helm/releases
>
> **resources/helm/helm-v2.13.1-linux-amd64.tar.gz**

```
# 放到k8s集群master节点

# 解压
tar -zxvf helm-v2.13.1-linux-amd64.tar.gz

# 复制helm二进制到 bin目录下，并且配置环境变量
cp linux-amd64/helm /usr/local/bin/

export PATH=$PATH:/usr/local/bin

# 查看是否安装成功
helm version
[
Client: &version.Version{SemVer:"v2.13.1",
GitCommit:"618447cbf203d147601b4b9bd7f8c37a5d39fbb4", GitTreeState:"clean"}
Error: could not find tiller

]
```

3.1.7 安装Tiller

> **重新安装tiller：helm reset -f**

> (1)安装tiller

```
helm init --upgrade -i registry.cn-
hangzhou.aliyuncs.com/google_containers/tiller:v2.13.1 --stable-repo-url
https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts

kubectl get pods -n kube-system -l app=helm
kubectl get svc -n kube-system -l app=helm
```

> (2)配置rbac

```
cat >helm-rbac-config.yaml<<EOF
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
EOF
kubectl create -f helm-rbac-config.yaml

# 配置tiller使用创建的ServiceAccount
kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":
{"template":{"spec":{"serviceAccount":"tiller"}}}}'
```

> (3)验证

```
# 查看pod启动情况
kubectl get pod -n kube-system -l app=helm

# 再次查看版本，显示出server版本
helm version
[
Client: &version.Version{SemVer:"v2.13.1",
GitCommit:"618447cbf203d147601b4b9bd7f8c37a5d39fbb4", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.13.1",
GitCommit:"618447cbf203d147601b4b9bd7f8c37a5d39fbb4", GitTreeState:"clean"}
]
```

3.1.8 使用helm操作chart

```
helm create:创建一个chart模板，比如：helm create test   --->ls test
helm package:打包一个chart模板，比如：helm package test  --->test-0.1.0.tgz
heml search:查找可用的chart模板，比如：helm search nginx
helm inspect:查看指定chart的基本信息，比如：helm inspect test
helm install:根据指定的chart部署一个release到k8s集群，比如：helm install test  ---
>get pods
```

3.1.9 chart模板

- chart文件结构

```
wordpress
├── charts                          # 存放chart的定义
├── Chart.yaml                      # 包含chart信息的yaml文件，如chart的版本、名称等
├── README.md                       # chart的介绍信息
├── requirements.lock
├── requirements.yaml               # chart需要的依赖
├── templates                       # k8s需要的资源
|   ├── deployment.yaml
|   ├── externaldb-secrets.yaml
|   ├── _helpers.tpl                # 存放可重用的模板片段
|   ├── ingress.yaml
|   ├── NOTES.txt
|   ├── pvc.yaml
|   ├── secrets.yaml
|   ├── svc.yaml
|   └── tls-secrets.yaml
└── values.yaml                      # 当前chart的默认配置的值
```

3.2 使用helm安装istio[自己探索]

> 官网：https://istio.io/docs/setup/install/helm/