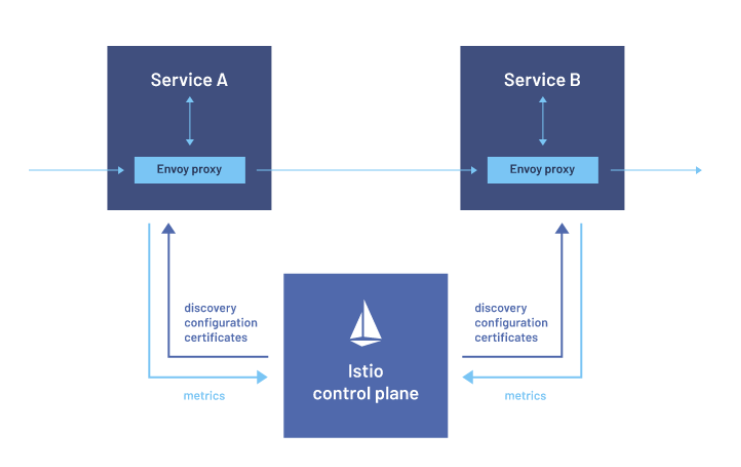
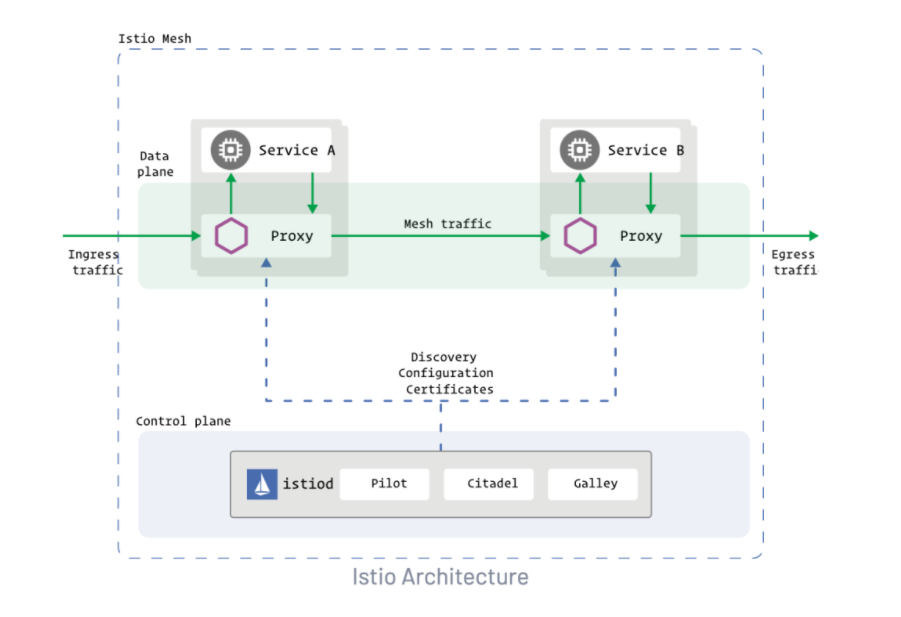
*The network should be transparent to applications. When network and application problems do occur it should be easy to determine the source of the problem.*

##### Istio 简介

service mesh 的提出是为了解决 k8s 中没有很好解决的服务治理部分内容。service mesh 的理念就是将服务与服务治理解耦；其中 service mesh 承担了服务治理的部分功能。service mesh 通过接管了集群服务的网络来实现负载均衡、熔断、流量控制等。

在service mesh 中，主流的解决方案是istio、linkerd；下文重点说明 istio；





istio 架构主要分为两部分：数据平面，控制平面；

###### istio 工作原理

在istio 与 k8s 结合的场景中，在minikube 集群安装 istio，会在k8s 中新建名为 istio-system 的 namespace；在此namespace 中启动了istio的控制平面；数据平面是基于envoy构建的代理、类似于pod中的pause 容器，通过注入实现和每个pod的一一对应，负责接管 pod的所有网络流量；类似于kubectl 工具，使用istioctl 工具将具体的服务治理配置提交给 istio 的控制平面，控制平面根据配置生成网络规则，下发给集群中的数据平面envoy代理，从而更新集群的服务规则。

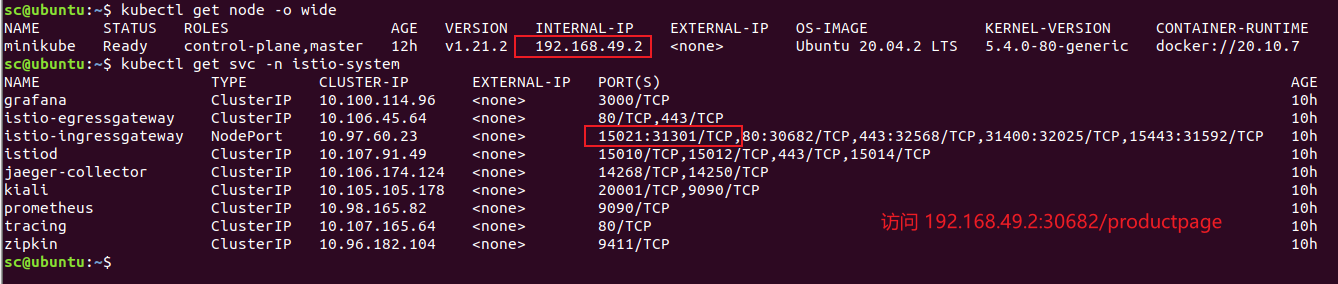
###### 控制平面原理

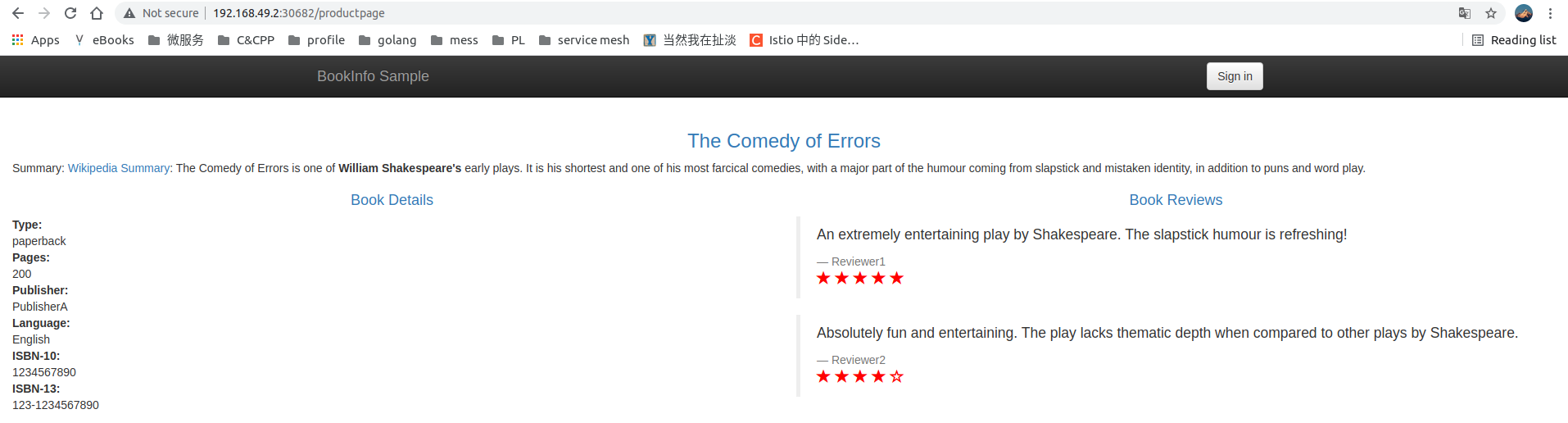
TODO

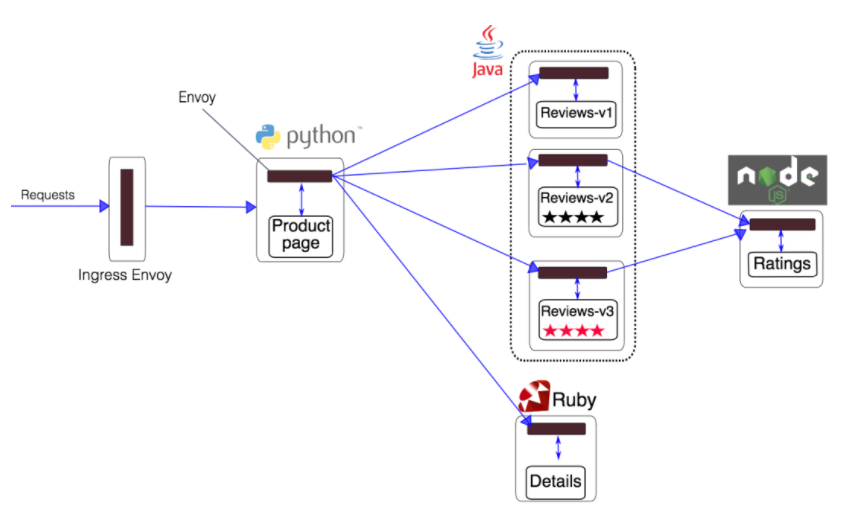
###### 数据平面原理

由上文可知，数据平面的主要工作是劫持 pod 与集群之间的流量，所以我们着重了解数据平面的envoy代理如何实现流量劫持；

首先在minikube 集群安装 istio；运行 istio 的例子项目 bookinfo；通过node\_ip:node\_port/productpage 可以访问到项目页面，多次刷新可以观察到其访问不同的reviews 服务。







###### sidecar 注入 istio 容器

在示例项目bookinfo.yaml 部署文件中的productpage部分只有一个镜像，就是业务容器；

...  
 spec:  
 serviceAccountName: bookinfo-details  
 containers:  
 - name: details  
 image: docker.io/istio/examples-bookinfo-details-v1:1.16.2  
 imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 9080  
 securityContext:  
 runAsUser: 1000  
...

在项目成功启动之后，使用kubectl describe 查看productpage 这个pod；

经过查看可知，该pod中的容器不止 productpage 容器；其中还包括istio-init、istio-proxy两个容器，其中istio-init容器运行结束后正常退出了；

kubectl describe pod productpage-v1-6b746f74dc-88nv2   
# 查看productpage pod的详细信息

istio 通过这种方式把数据平面注入到服务中；其中istio-init 容器设置网络命名空间的iptables 中 nat 表规则；istio-proxy 负责接收来自控制平面的分发规则， 负责复杂的网络控制；

###### istio-init 做了什么

通过describe pod 可知，运行istio-init 容器的命令是：

istio-iptables -p 15001 -z 15006 -u 1337 -m REDIRECT -i '\*' -x "" -b '\*' -d 15090,15020

运行的 istio-init 容器做的事情就是根据命令行启动参数来设置 iptables 规则；通过 iptables 劫持pod中的流量，

1. minikube ssh 登录到 k8s 的node，并且查看productpage 容器的ID；

* minikube ssh  
  docker ps | grep productpage
* 看到有三个容器，分别是productpage、istio-proxy、pause三个容器；

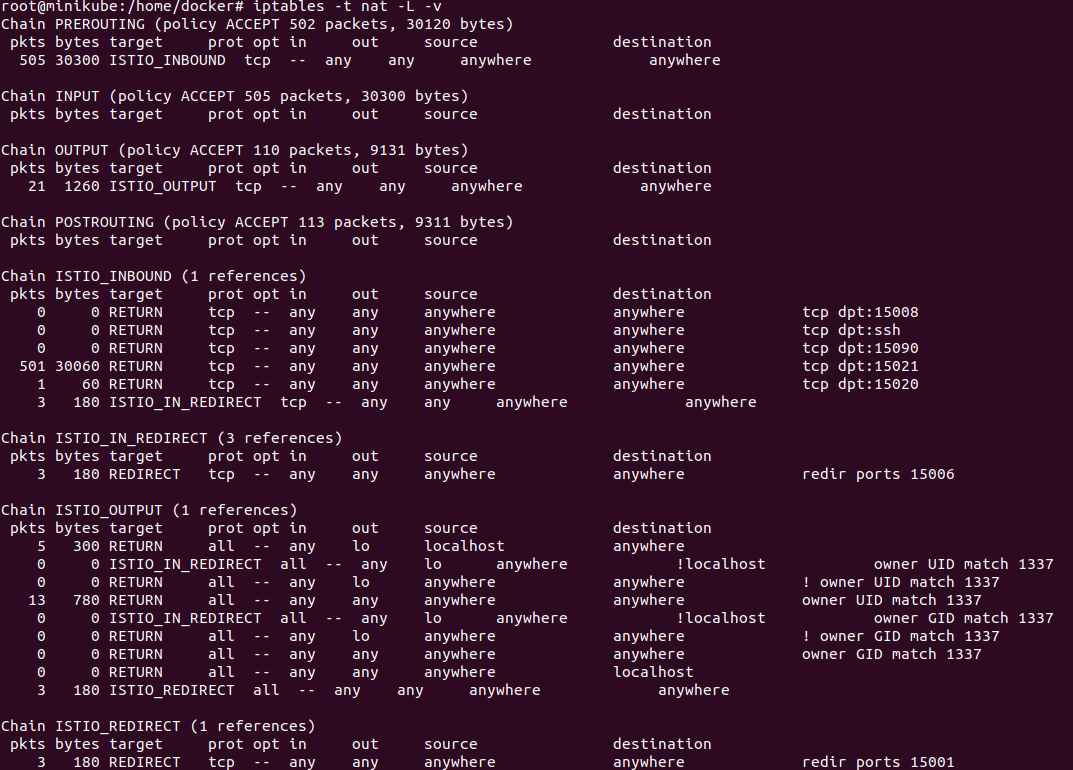
1. 根据容器ID，找到容器中运行的进程在node节点上的PID

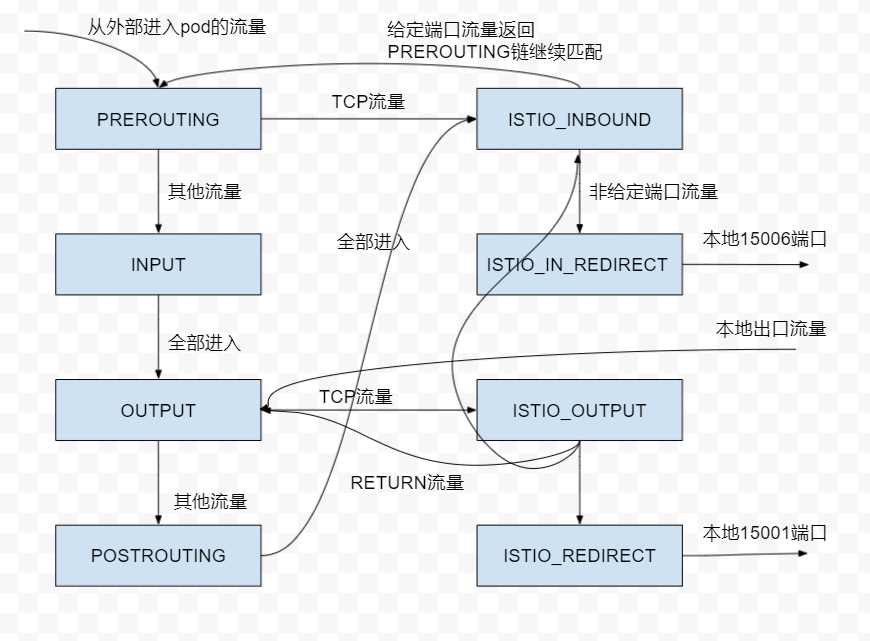
docker top 4da41145304f

1. 根据找到的PID，进入与PID相同的namespace（通过这种方式进入namespace，避免了缺少查看信息工具的问题）

* sudo nsenter -n -t 6637

1. 查看此namespace的iptables规则

* iptables -t nat -L -v
* 
* 由 iptables 中 nat 表可知，nat 表有PREROUTING、INPUT、OUTPUT、POSTROUTING、ISTIO\_INBOUND、ISTIO\_IN\_REDIRECT、ISTIO\_OUTPUT 几个规则链，经过分析可知：
  + 流入 PREROUTING 链的TCP流量全部转入 ISTIO\_INBOUND 链；
  + 流入 PREROUTING 链的其他流量依次流入INPUT链；
  + INPUT链没有规则，则依次流入OUTOUT链；
  + 流入OUTPUT链的TCP流量全部转入ISTIO\_OUTPUT链；
  + 流入OUTPUT链的其他流量依次流入POSTROUTING链；
  + POSTROUTING链没有定义规则，则会依次流入ISTIO\_INBOUND链；
  + 流入ISTIO\_INBOUND链的dpt 为若干端口的流量处理方法是RETURE，即返回给流入ISTIO\_INBOUND的链继续匹配规则；不在这些端口中的流量会转入ISTIO\_IN\_REDIRECT链中；
  + 流入ISTIO\_IN\_REDIRECT链的流量被转发到本地15006端口；
  + 流入ISTIO\_OUTPUT链的流量根据规则，转到RETURN、ISTIO\_IN\_REDIRECT、ISTIO\_REDIRECT链；
  + 最后流入ISTIO\_REDIRECT链的流量被转发到本地15001端口



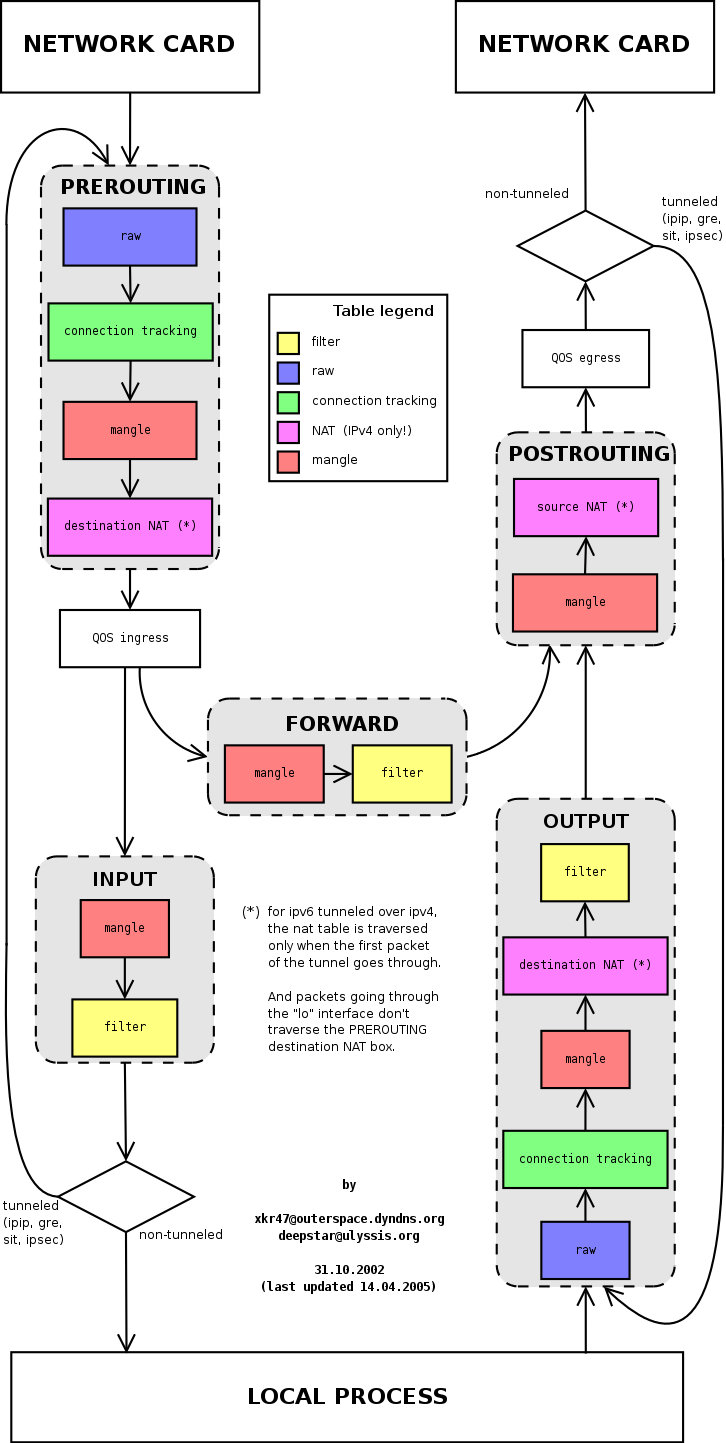
**小结**：istio-init 容器实际是运行了 istio-iptables 这个镜像，通过运行参数设置劫持pod流量，init 容器做的事就是增加pod网络命名空间中的iptables规则。劫持从本地发送出来的数据到本地15001端口，劫持从外界发送到pod的流量到15006端口。同时明白了 istio-proxy 容器的作用，肯定是监听本地15001和15006端口，负责更高层的流量管理；

###### istio-proxy 做了什么

TODO

##### 附

###### iptables



iptables