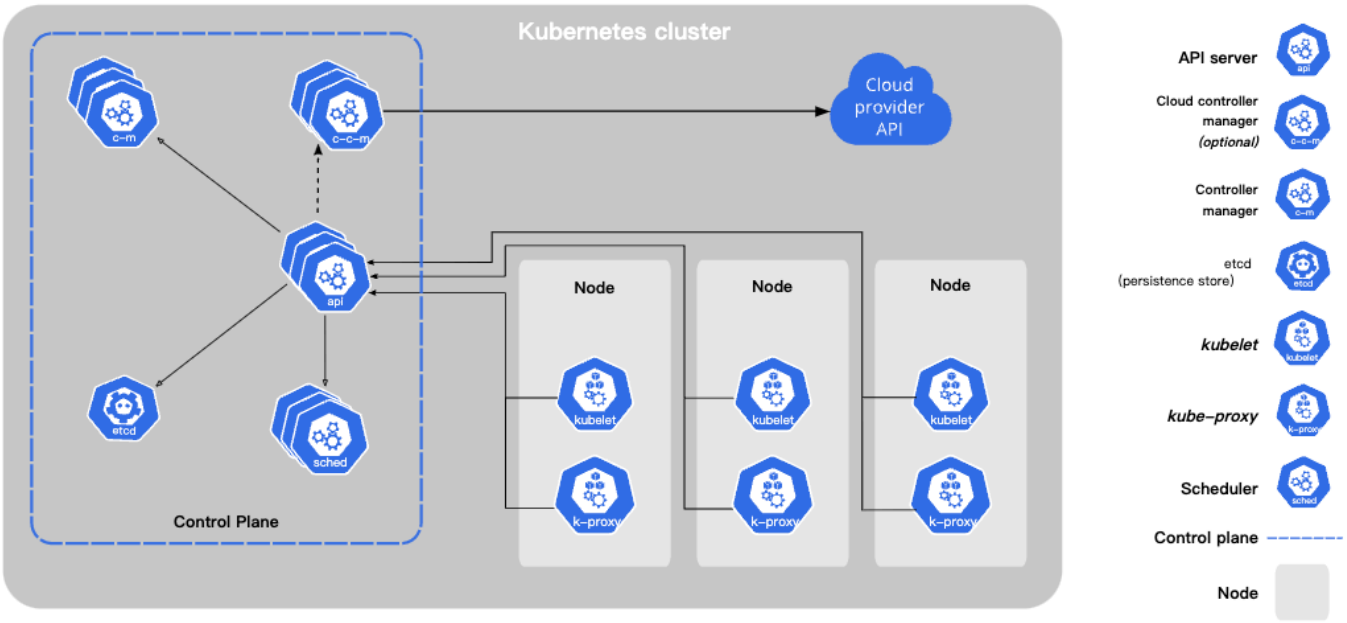
Kubernetes安全

Kubernetes是什么?



Kubernetes是容器集群管理系统，是一个开源的平台，可以实现容器集群的自动化部署、自动扩缩容、维护等功能。

通过Kubernetes你可以：

- 快速部署应用
- 快速扩展应用
- 无缝对接新的应用功能
- 节省资源，优化硬件资源的使用

我们的目标是促进完善组件和工具的生态系统，以减轻应用程序在公有云或私有云中运行的负担。

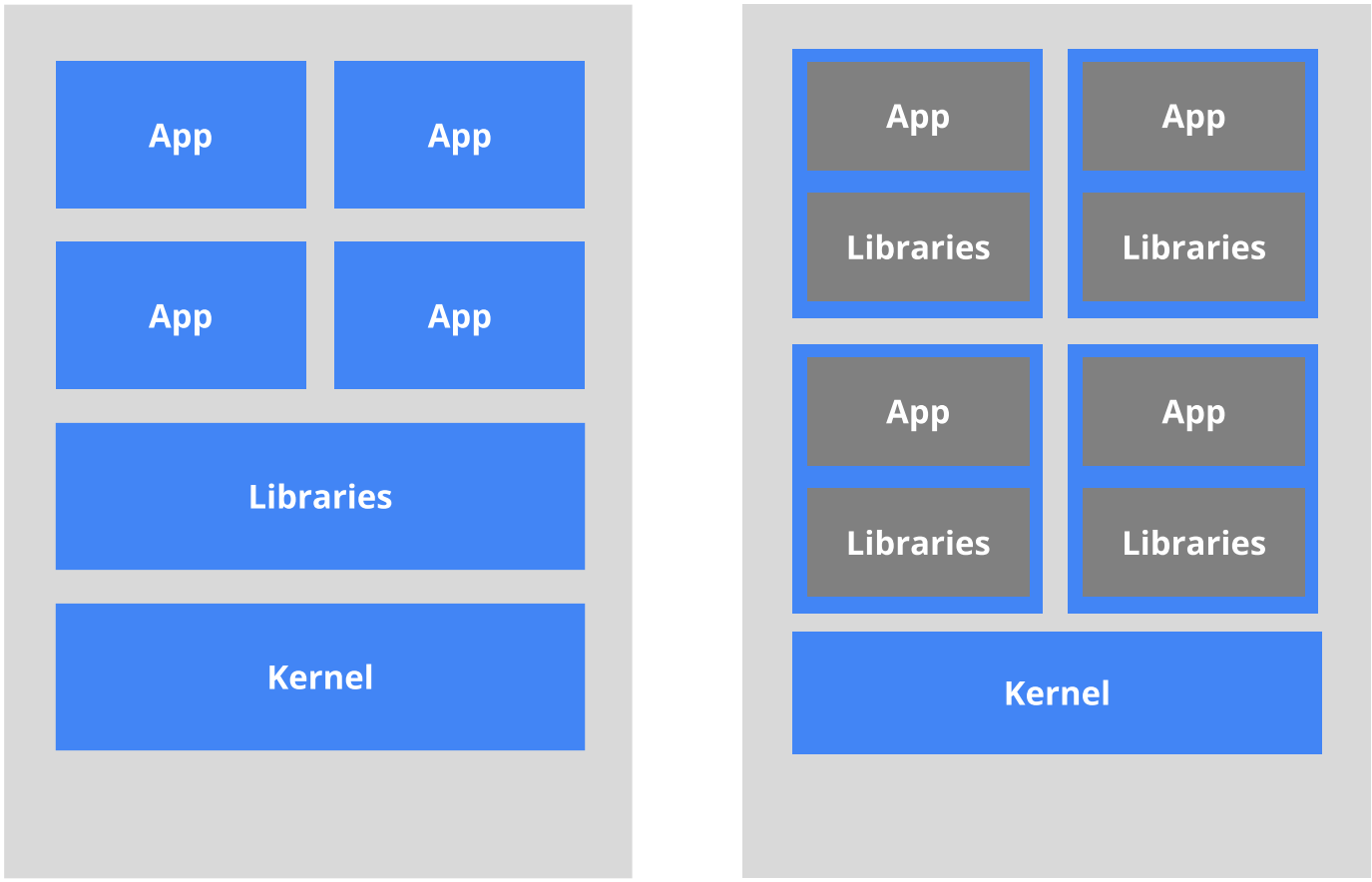
Kubernetes 特点

- **可移植**: 支持公有云，私有云，混合云，多重云（multi-cloud）
- **可扩展**: 模块化, 插件化, 可挂载, 可组合
- **自动化**: 自动部署，自动重启，自动复制，自动伸缩/扩展

Kubernetes是Google 2014年创建管理的，是Google 10多年大规模容器管理技术Borg的开源版本。

为什么要使用容器?

为什么要使用容器？通过以下两个图对比：



传统的应用部署方式是通过插件或脚本来安装应用。这样做的缺点是应用的运行、配置、管理、所有生存周期将与当前操作系统绑定，这样做并不利于应用的升级更新/回滚等操作，当然也可以通过创建虚机的方式来实现某些功能，但是虚拟机非常重，并不利于可移植性。

新的方式是通过部署容器方式实现，每个容器之间互相隔离，**每个容器有自己的文件系统** ，容器之间进程不会相互影响，能区分计算资源。相对于虚拟机，容器能快速部署，由于容器与底层设施、机器文件系统解耦的，所以它能在不同云、不同版本操作系统间进行迁移。

容器占用资源少、部署快，每个应用可以被打包成一个容器镜像，每个应用与容器间成一对一关系也使容器有更大优势，使用容器可以在build或release 的阶段，为应用创建容器镜像，因为每个应用不需要与其余的应用堆栈组合，也不依赖于生产环境基础结构，这使得从研发到测试、生产能提供一致环境。类似地，容器比虚机轻量、更“透明”，这更便于监控和管理。最后，

容器优势总结：

- **快速创建/部署应用：**与VM虚拟机相比，容器镜像的创建更加容易。
- **持续开发、集成和部署：**提供可靠且频繁的容器镜像构建/部署，并使用快速和简单的回滚(由于镜像不可变性)。
- **开发和运行相分离：**在build或者release阶段创建容器镜像，使得应用和基础设施解耦。
- **开发，测试和生产环境一致性：**在本地或外网（生产环境）运行的一致性。
- **云平台或其他操作系统：**可以在 Ubuntu、RHEL、 CoreOS、on-prem、Google Container Engine或其它任何环境中运行。
- **Loosely coupled，分布式，弹性，微服务化：**应用程序分为更小的、独立的部件，可以动态部署和管理。

- **资源隔离**
- **资源利用**：更高效

使用Kubernetes能做什么?

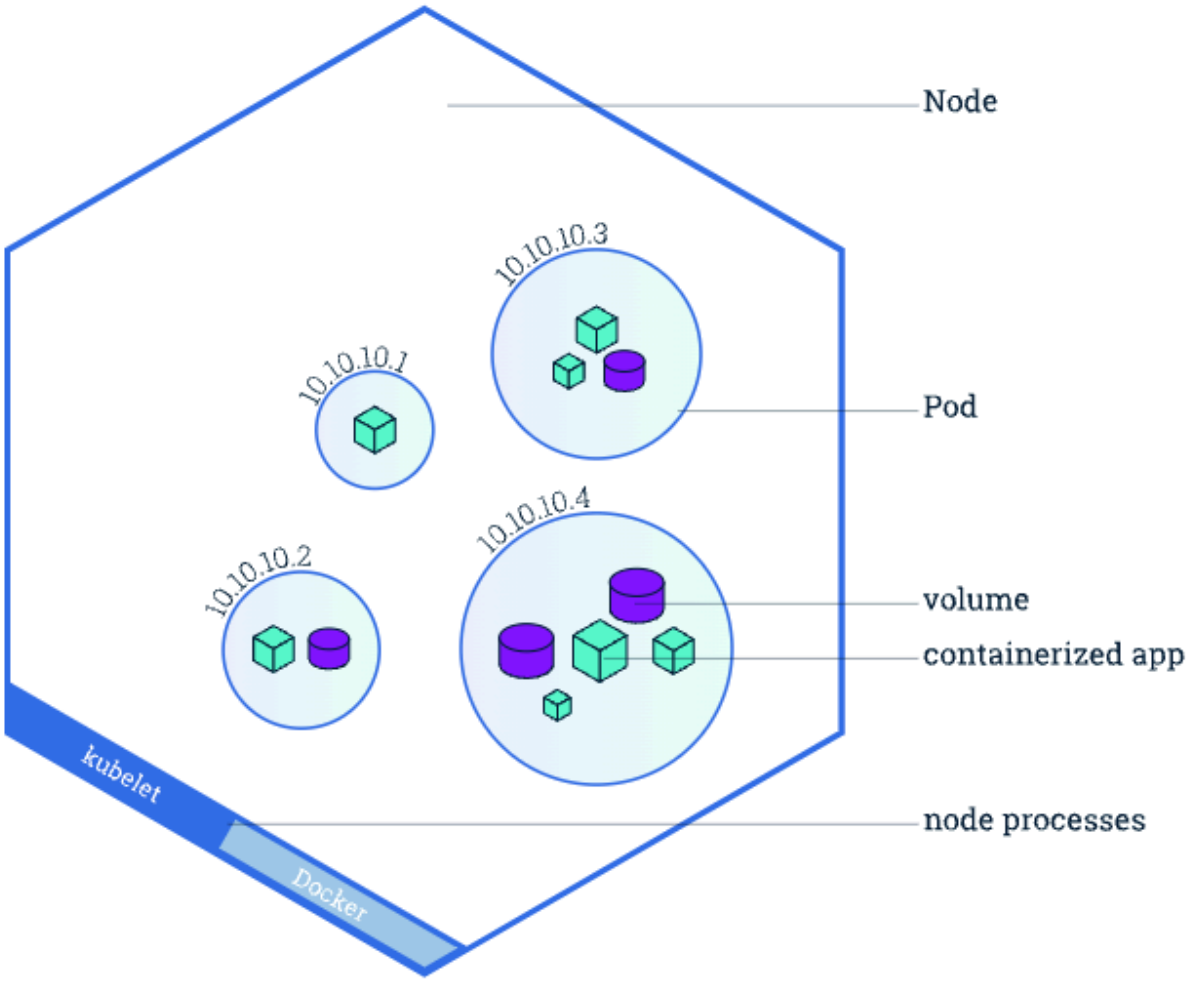
可以在物理或虚拟机的Kubernetes集群上运行容器化应用，Kubernetes能提供一个以“**容器为中心的基础架构**”，满足在生产环境中运行应用的一些常见需求，如：

- 多个进程（作为容器运行）协同工作Pod）
- 存储系统挂载
- Distributing secrets(分发凭证)
- 应用健康检测
- 应用实例的复制
- Pod自动伸缩/扩展
- Naming and discovering(命名和发现)
- 负载均衡
- 滚动更新
- 资源监控
- 日志访问
- 调试应用程序
- 提供认证和授权

*Kubernetes*是什么意思？K8S?

Kubernetes的名字来自希腊语，意思是“*舵手*” 或 “*领航员*”。*K8s*是将8个字母“ubernete”替换为“8”的缩写。

Kubernetes 组件



Master 组件

Master组件提供集群的管理控制中心。

Master组件可以在集群中任何节点上运行。但是为了简单起见，通常在一台VM/机器上启动所有Master组件，并且不会在此VM/机器上运行用户容器。

kube-apiserver

kube-apiserver用于暴露Kubernetes API。任何的资源请求/调用操作都是通过kube-apiserver提供的接口进行。

ETCD

etcd是Kubernetes提供默认的存储系统，保存所有集群数据，使用时需要为etcd数据提供备份计划。

kube-controller-manager

kube-controller-manager运行管理控制器，它们是集群中处理常规任务的后台线程。逻辑上，每个控制器是一个单独的进程，但为了降低复杂性，它们都被编译成单个二进制文件，并在单个进程中运行。

这些控制器包括：

- 节点（Node）控制器。

- 副本（Replication）控制器：负责维护系统中每个副本中的pod。
- 端点（Endpoints）控制器：填充Endpoints对象（即连接Services＆Pods）。
- Service Account和Token控制器：为新的Namespace创建默认帐户访问API Token。

cloud-controller-manager

云控制器管理器负责与底层云提供商的平台交互。

云控制器管理器仅运行云提供商特定的（controller loops）控制器循环。可以通过将`--cloud-provider` flag设置为external启动kube-controller-manager ，来禁用控制器循环。

cloud-controller-manager 具体功能：

- 节点（Node）控制器
- 路由（Route）控制器
- Service控制器
- 卷（Volume）控制器

kube-scheduler

kube-scheduler 监视新创建没有分配到Node的Pod，为Pod选择一个Node。

插件 addons

插件（addon）是实现集群pod和Services功能的。

DNS

虽然不严格要求使用插件，但Kubernetes集群都应该具有集群 DNS。

群集 DNS是一个DNS服务器，能够为 Kubernetes services提供 DNS记录。

由Kubernetes启动的容器自动将这个DNS服务器包含在他们的DNS searches中。

用户界面

kube-ui提供集群状态基础信息查看。

容器资源监测

容器资源监控提供一个UI浏览监控数据。

Cluster-level Logging

负责保存容器日志，搜索/查看日志。

节点（Node）组件

Kubernetes 通过将容器放入在节点（Node）上运行的 Pod 中来执行你的工作负载。 节点可以是一个虚拟机或者物理机器，取决于所在的集群配置。 每个节点包含运行 Pods所需的服务； 这些节点由 控制面负责管理。

通常集群中会有若干个节点；而在一个学习用或者资源受限的环境中，你的集群中也可能 只有一个节点。

节点上的组件包括 kubelet、container runtime 以及 kube-proxy。

kubelet

kubelet是主要的节点代理，它会监视已分配给节点的pod，具体功能：

- 安装Pod所需的volume。
- 下载Pod的Secrets。
- Pod中运行的 docker（或rkt）容器。
- 定期执行容器健康检查。
- 如有必要，通过创建一个“镜像pod”，将pod的状态报告给系统的其余部分。
- 向系统的其余部分报告节点的状态。

kube-proxy

kube-proxy通过在主机上维护网络规则并执行连接转发来实现Kubernetes服务抽象。

docker

docker用于运行容器。

RKT

rkt运行容器，作为docker工具的替代方案。

supervisord

supervisord是一个轻量级的监控系统，用于保障kubelet和docker运行。

fluentd

fluentd是一个守护进程，可提供cluster-level logging。

> Kubernetes Deployment 是检查Pod的健康状况，如果它终止，则重新启动一个Pod的容器，Deployment管理Pod的创建和扩展。

了解Kubernetes对象

了解Kubernetes对象

Kubernetes对象是Kubernetes系统中的持久实体。Kubernetes使用这些实体来表示集群的状态。具体来说，他们可以描述：

- 容器化应用正在运行(以及在哪些节点上)
- 这些应用可用的资源
- 关于这些应用如何运行的策略，如重新策略，升级和容错

Kubernetes对象是“record of intent”，一旦创建了对象，Kubernetes系统会确保对象存在。通过创建对象，可以有效地告诉Kubernetes系统你希望集群的工作负载是什么样的。

要使用Kubernetes对象（无论是创建，修改还是删除），都需要使用Kubernetes API。例如，当使用kubectl命令管理工具时，CLI会为提供Kubernetes API调用。你也可以直接在自己的程序中使用Kubernetes API，Kubernetes提供一个golang客户端库。

对象（Object）规范和状态

每个Kubernetes对象都包含两个嵌套对象字段，用于管理Object的配置：Object Spec和Object Status。Spec描述了对象所需的状态 - 希望Object具有的特性，Status描述了对象的实际状态，并由Kubernetes系统提供和更新。

例如，通过Kubernetes Deployment 来表示在集群上运行的应用的对象。创建Deployment时，可以设置Deployment Spec，来指定要运行应用的三个副本。Kubernetes系统将读取Deployment Spec，并启动你想要的三个应用实例 - 来更新状态以符合之前设置的Spec。如果这些实例中有任何一个失败（状态更改），Kuberentes系统将响应Spec和当前状态之间差异来调整，这种情况下，将会开始替代实例。

> Kubernetes Deployment 是检查Pod的健康状况，如果它终止，则重新启动一个Pod的容器，Deployment管理Pod的创建和扩展。

描述Kubernetes对象

在Kubernetes中创建对象时，必须提供描述其所需Status的对象Spec，以及关于对象（如name）的一些基本信息。当使用Kubernetes API创建对象（直接或通过kubectl）时，该API请求必须将该信息作为JSON包含在请求body中。通常，可以将信息提供给kubectl .yaml文件，在进行API请求时，kubectl将信息转换为JSON。

以下示例是一个.yaml文件，显示Kubernetes Deployment所需的字段和对象Spec：

nginx-deployment.yaml

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

使用上述.yaml文件创建Deployment，是通过在kubectl中使[kubectl create]命令来实现。将该.yaml文件作为参数传递。如下例子：

```
$ kubectl create -f docs/user-guide/nginx-deployment.yaml --record
```

其输出与此类似:

```
deployment "nginx-deployment" created
```

必填字段

对于要创建的Kubernetes对象的yaml文件，需要为以下字段设置值：

- apiVersion - 创建对象的Kubernetes API 版本
- kind - 要创建什么样的对象?
- metadata- 具有唯一标示对象的数据，包括 name（字符串）、UID和Namespace（可选项）

还需要提供对象Spec字段，对象Spec的精确格式（对于每个Kubernetes 对象都是不同的），以及容器内嵌套的特定于该对象的字段。

Kubernetes Namespaces

Kubernetes可以使用Namespaces（命名空间）创建多个虚拟集群。

何时使用多个Namespaces

当团队或项目中具有许多用户时，可以考虑使用Namespace来区分，a如果是少量用户集群，可以不需要考虑使用Namespace，如果需要它们提供特殊性质时，可以开始使用Namespace。

Namespace为名称提供了一个范围。资源的Names在Namespace中具有唯一性。

Namespace是一种将集群资源划分为多个用途(通过 resource quota)的方法。

在未来的Kubernetes版本中，默认情况下，相同Namespace中的对象将具有相同的访问控制策略。

对于稍微不同的资源没必要使用多个Namespace来划分，例如同意软件的不同版本，可以使用labels(标签)来区分同一Namespace中的资源。

使用 Namespaces

Namespace的创建、删除和查看。

创建

```
(1) 命令行直接创建
$ kubectl create namespace new-namespace

(2) 通过文件创建
$ cat my-namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: new-namespace

$ kubectl create -f ./my-namespace.yaml
```

注意：命名空间名称满足正则表达式a-z0-9?,最大长度为63位

删除

```
$ kubectl delete namespaces new-namespace
```

注意：

1. 删除一个namespace会自动删除所有属于该namespace的资源。
2. default和kube-system命名空间不可删除。
3. PersistentVolumes是不属于任何namespace的，但PersistentVolumeClaim是属于某个特定namespace的。
4. Events是否属于namespace取决于产生events的对象。

查看 Namespaces

使用以下命令列出群集中的当前的Namespace：

```
$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    1d
kube-system   Active    1d
```

Kubernetes从两个初始的Namespace开始：

- default
- kube-system 由Kubernetes系统创建的对象的Namespace

通过请求设置namespace

要临时设置Request的Namespace，请使用--namespace 标志。

例如：

```
$ kubectl --namespace=<insert-namespace-name-here> run nginx --image=nginx
$ kubectl --namespace=<insert-namespace-name-here> get pods
```

Kubernetes Nodes

Node是什么?

Node是Kubernetes中的工作节点，最开始被称为minion。一个Node可以是VM或物理机。每个Node（节点）具有运行pod的一些必要服务，并由Master组件进行管理，Node节点上的服务包括Docker、kubelet和kube-proxy。有关更多详细信息，请参考架构设计文档中的"Kubernetes Node"部分。

Node Status

节点的状态信息包含：

- Addresses
- ~~Phase~~ (已弃用)
- Condition
- Capacity
- Info

下面详细描述每个部分。

Addresses

这些字段的使用取决于云提供商或裸机配置。

- HostName：可以通过kubelet 中 --hostname-override参数覆盖。
- ExternalIP：可以被集群外部路由到的IP。
- InternalIP：只能在集群内进行路由的节点的IP地址。

Phase

不推荐使用，已弃用。

Condition

conditions字段描述所有Running节点的状态。

Node Condition	Description
OutOfDisk	True：如果节点上没有足够的可用空间来添加新的pod；否则为：False
Ready	True：如果节点是健康的并准备好接收pod；False：如果节点不健康并且不接受pod；Unknown：如果节点控制器在过去40秒内没有收到node的状态报告。
MemoryPressure	True：如果节点存储器上内存过低; 否则为：False。
DiskPressure	True：如果磁盘容量存在压力 - 也就是说磁盘容量低；否则为：False。

node condition被表示为一个JSON对象。例如，下面的响应描述了一个健康的节点。

```
"conditions": [
  {
    "kind": "Ready",
    "status": "True"
  }
]
```

如果Ready condition的Status是“Unknown” 或 “False”，比“pod-eviction-timeout”的时间长，则传递给“ kube-controller-manager”的参数，该节点上的所有Pod都将被节点控制器删除。默认的eviction timeout时间为5分钟。在某些情况下，当节点无法访问时，apiserver将无法与kubelet通信，删除Pod的需求不会传递到kubelet，直到重新与apiserver建立通信，这种情况下，计划删除的Pod会继续在划分的节点上运行。

在Kubernetes 1.5之前的版本中，节点控制器将强制从apiserver中删除这些不可达（上述情况）的pod。但是，在1.5及更高版本中，节点控制器在确认它们已经停止在集群中运行之前，不会强制删除Pod。可以看到这些可能在不可达节点上运行的pod处于"Terminating"或 “Unknown”。如果节点永久退出集群，Kubernetes是无法从底层基础架构辨别出来，则集群管理员需要手动删除节点对象，从Kubernetes删除节点对象会导致运行在上面的所有Pod对象从apiserver中删除，最终将会释放names。

Capacity

描述节点上可用的资源：CPU、内存和可以调度到节点上的最大pod数。

Info

关于节点的一些基础信息，如内核版本、Kubernetes版本（kubelet和kube-proxy版本）、Docker版本（如果有使用）、OS名称等。信息由Kubelet从节点收集。

Management

与 pods 和 services 不同，**节点不是由Kubernetes 系统创建，它是由Google Compute Engine等云提供商在外部创建的，或使用物理和虚拟机**。这意味着当Kubernetes创建一个节点时，它只是创建一个代表节点的对象，创建后，Kubernetes将检查节点是否有效。例如，如果使用以下内容创建一个节点：

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "10.240.79.157",
    "labels": {
      "name": "my-first-k8s-node"
    }
  }
}
```

Kubernetes将在内部创建一个节点对象，并通过基于metadata.name字段的健康检查来验证节点，如果节点有效，即所有必需的服务会同步运行，则才能在上面运行pod。请注意，Kubernetes将保留无效节点的对象（除非客户端有明确删除它）并且它将继续检查它是否变为有效。

目前，有三个组件与Kubernetes节点接口进行交互：节点控制器（node controller）、kubelet和kubectl。

Node Controller

节点控制器（Node Controller）是管理节点的Kubernetes master组件。

Kubernetes Pod概述

Kubernetes对象模型中可部署的最小对象。

了解Pod

Pod是Kubernetes创建或部署的最小/最简单的基本单位，一个Pod代表集群上正在运行的一个进程。

一个Pod封装一个应用容器（也可以有多个容器），存储资源、一个独立的网络IP以及管理控制容器运行方式的策略选项。Pod代表部署的一个单位：Kubernetes中单个应用的实例，它可能由单个容器或多个容器共享组成的资源。

> Docker是Kubernetes Pod中最常见的runtime，Pods也支持其他容器runtimes。

Kubernetes中的Pod使用可分两种主要方式：

- Pod中运行一个容器。“one-container-per-Pod”模式是Kubernetes最常见的用法; 在这种情况下，你可以将Pod视为单个封装的容器，但是Kubernetes是直接管理Pod而不是容器。
- Pods中运行多个需要一起工作的容器。Pod可以封装紧密耦合的应用，它们需要由多个容器组成，它们之间能够共享资源，这些容器可以形成一个单一的内部service单位 - 一个容器共享文件，另一个“sidecar”容器来更新这些文件。Pod将这些容器的存储资源作为一个实体来管理。

每个Pod都是运行应用的单个实例，如果需要水平扩展应用（例如，运行多个实例），则应该使用多个Pods，每个实例一个Pod。在Kubernetes中，这样通常称为Replication。Replication的Pod通常由Controller创建和管理。

Ingress

Ingress 是对集群中服务的外部访问进行管理的 API 对象，典型的访问方式是 HTTP。

Ingress 可以提供负载均衡、SSL 终结和基于名称的虚拟托管。

术语

- 节点（Node）：Kubernetes 集群中的一台工作机器，是集群的一部分。
- 集群（Cluster）：一组运行由 Kubernetes 管理的容器化应用程序的节点。 在此示例和在大多数常见的 Kubernetes 部署环境中，集群中的节点都不在公共网络中。
- 边缘路由器（Edge Router）：在集群中强制执行防火墙策略的路由器。可以是由云提供商管理的网关，也可以是物理硬件。
- 集群网络（Cluster Network）：一组逻辑的或物理的连接，根据 Kubernetes 网络模型在集群内实现通信。
- 服务（Service）： Kubernetes 服务（Service）， 使用标签选择器（selectors）辨认一组 Pod。 除非另有说明，否则假定服务只具有在集群网络中可路由的虚拟 IP。

Ingress 是什么?

Ingress公开了从集群外部到集群内服务的 HTTP 和 HTTPS 路由。 流量路由由 Ingress 资源上定义的规则控制。

下面是一个将所有流量都发送到同一 Service 的简单 Ingress 示例：

clusterIngress-管理的
负载均衡器路由规则IngressPodServicePod客户端

Ingress 可为 Service 提供外部可访问的 URL、负载均衡流量、终止 SSL/TLS，以及基于名称的虚拟托管。 Ingress 控制器 通常负责通过负载均衡器来实现 Ingress，尽管它也可以配置边缘路由器或其他前端来帮助处理流量。

Ingress 不会公开任意端口或协议。 将 HTTP 和 HTTPS 以外的服务公开到 Internet 时，通常使用 Service.Type=NodePort或Service.Type=LoadBalancer类型的 Service。

搭建 Kubernetes

Docker for Mac ， Docker for Windows开启 Kubernetes

- 安装 Docker Desktop

Docker Desktop下载地址

- 为 Docker daemon 配置 Docker Hub 的中国官方镜像加速 `https://registry.docker-cn.com`

> 由于Kubernetes大量的容器镜像在 gcr.io， 无法在国内保证稳定的访问。我们提供了一些工具脚本，帮助从阿里云镜像服务下载所需镜像

```
git clone https://github.com/AliyunContainerService/k8s-for-docker-desktop
cd k8s-for-docker-desktop
```

预先从阿里云Docker镜像服务下载 Kubernetes 所需要的镜像, 可以通过修改 `images.properties` 文件加载你自己需要的镜像

```
./load_images.sh
```

Windows打开Powershell执行

```
.\load_images.ps1
```

安装 kubectl

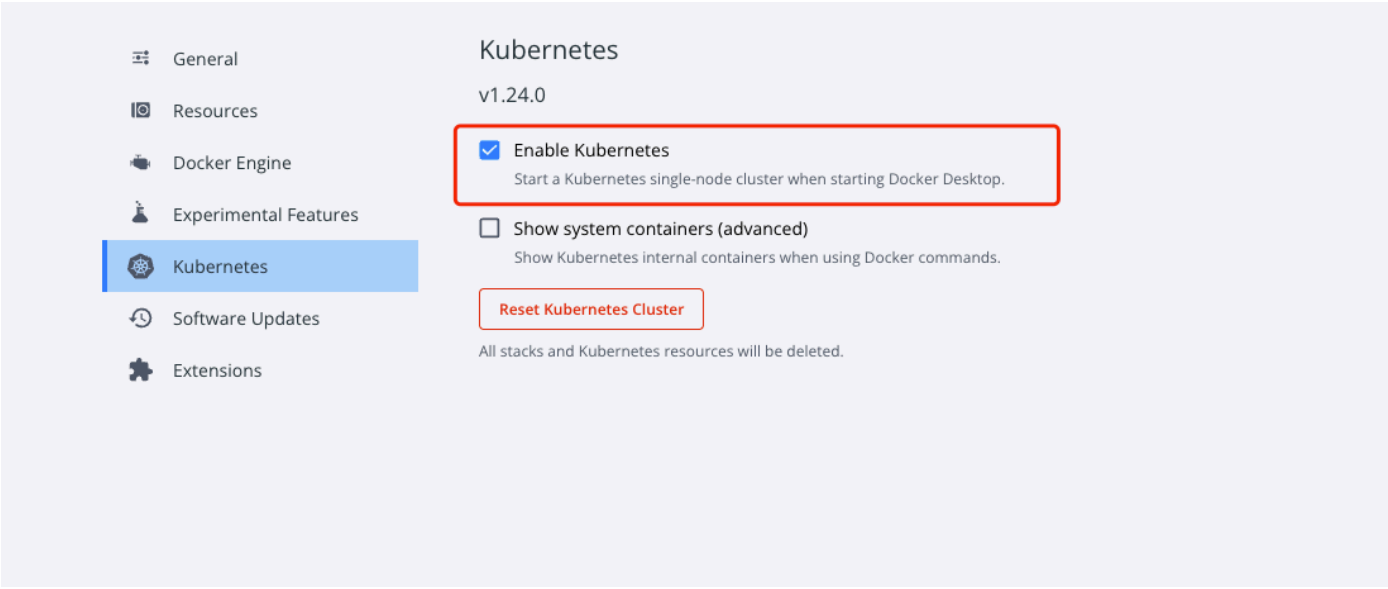
- 在 Linux 上安装 kubectl
- 在 macOS 上安装 kubectl
- 在 Windows 上安装 kubectl

基础命令

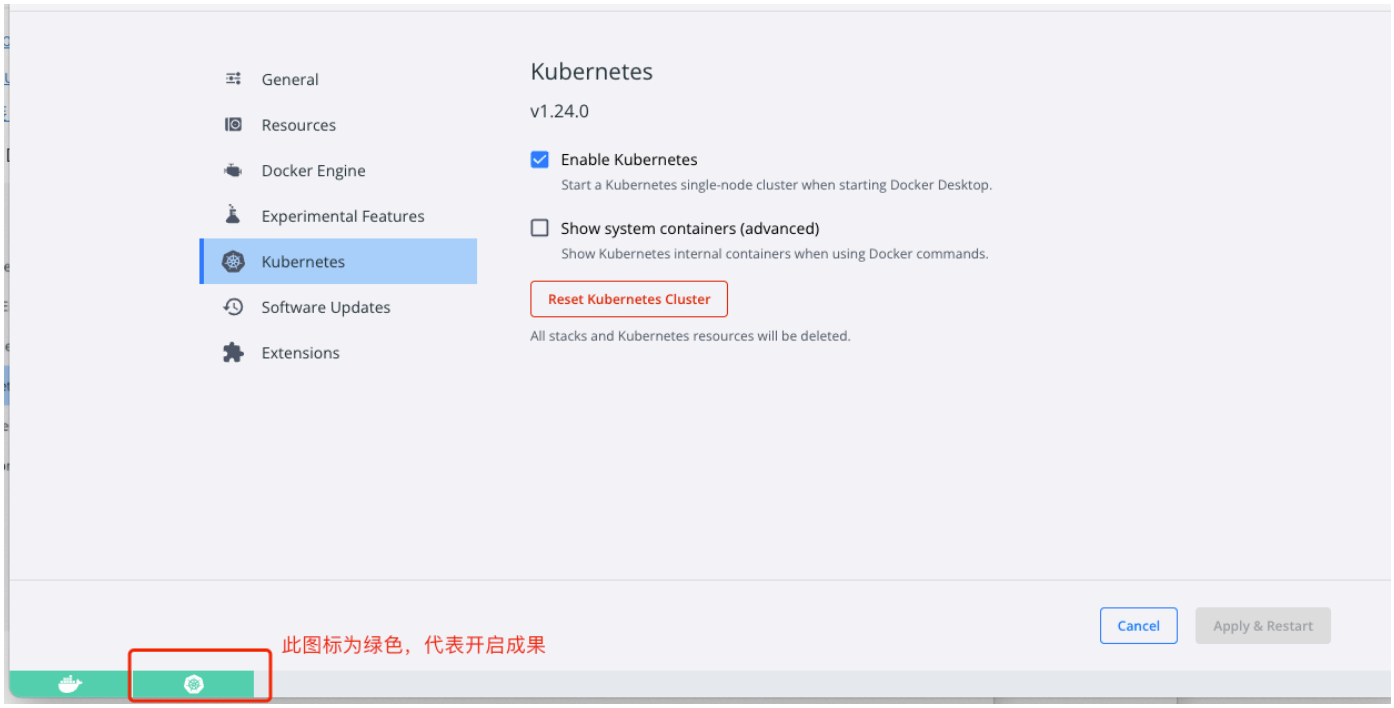
```
# 基于yaml文件创建namespace
kubectl create -f test.yaml
# 命令行创建namespace
kubectl create nampspace custom-namespace
# 列出所有命名空间
kubectl get ns
kubectl get namespace
# 列出命名空间下的资源,不指定命名空间的情况下默认为default命名空间
kubectl get pods -n kube-system
```

安装过程

安装好之后，在Docker Desktop的设置中开启 kubernetes，选择 `Enable Kubernetes`



通过图标确认是否开启成功



启动成功后，通过`kubectl cluster-info`查看状态

```
> kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

创建 kubernetes-dashboard

```
cd k8s-for-docker-desktop
kubectl create -f kubernetes-dashboard.yaml

#### 配置控制台访问令牌

授权`kube-system`默认服务账号

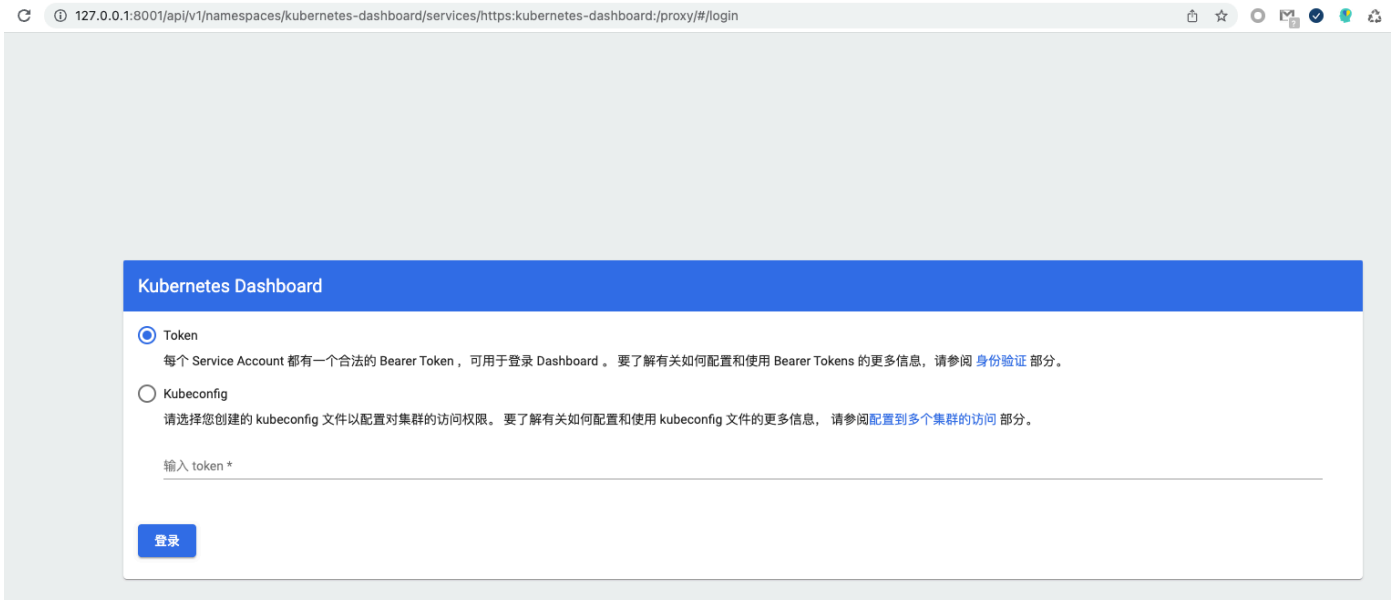
```shell
kubectl apply -f kube-system-default.yaml
```
```

创建成功后执行命令,开启转发

```
kubectl proxy
```

访问链接

`http://127.0.0.1:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy`



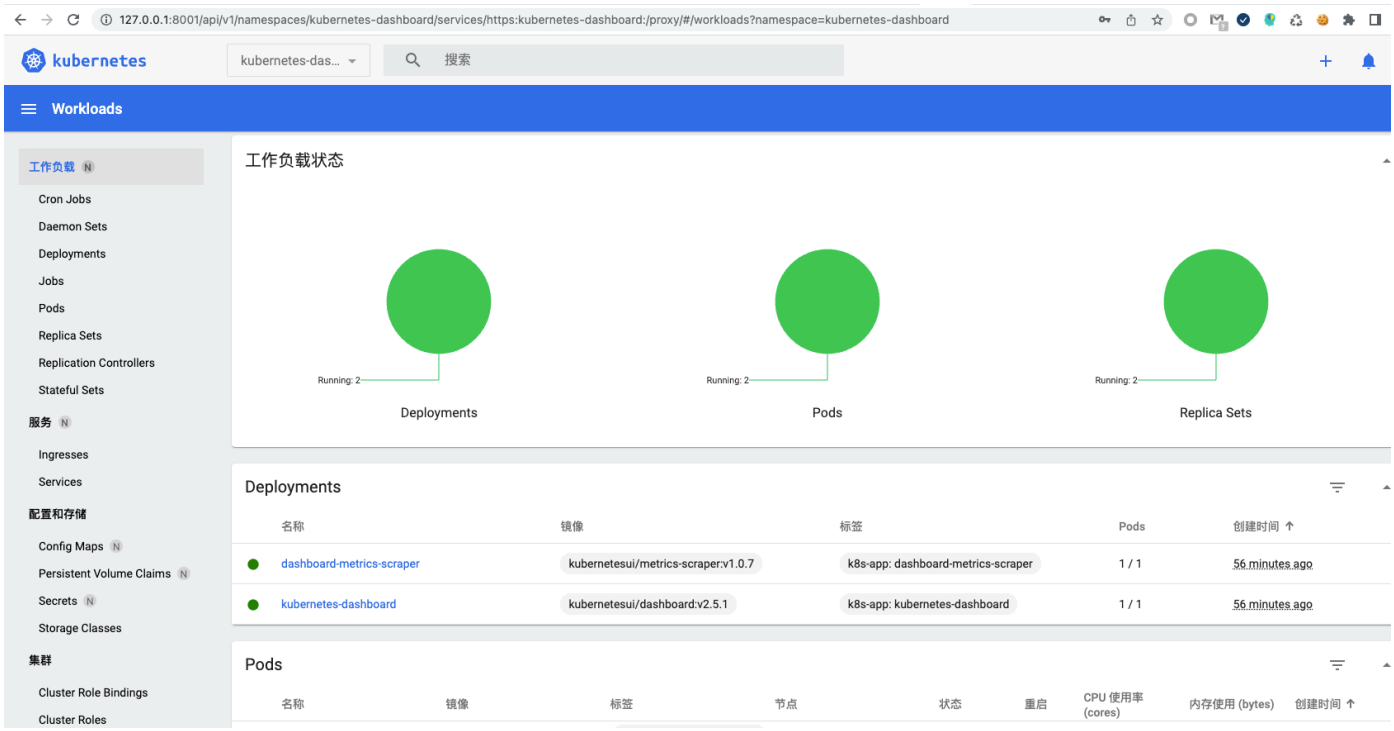
选择`Token`,在命令行输入命令 `kubectl describe secret default -n kube-system`

```
> kubectl describe secret default -n kube-system
Name:         default
Namespace:    kube-system
Labels:       k8s-app=kube-system
Annotations:  kubernetes.io/service-account.name: default
              kubernetes.io/service-account.uid: f57bcf28-d425-45aa-995c-194114dc00cf

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1099 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IlM0OVdoV0h0MkYxNTMxNzhVakFURVVIUVVGTERvckJya1J6WWtXbktueG8ifQ.eyJpc3MiOiJrdWJlc
m5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMua
W8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZ
SI6ImRlZmF1bHQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJmNTdiY2YyOC1kNDI1LTQ1YWEtOTk1Yy0xO
TQxMTRkYzAwY2YiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06ZGVmYXVsdCJ9.p5NfvtJ0xUILRFImegJ93-3SbhEzwvLWUI7M
i4P4u0qfMOHa205rYIG-NFh2rrrSEWBmztoCNx-DGfnuo30qerY0x2SgFey7Nx252PQP19bL3IrgWImMLBORUL6DP8xHwUuTLlkDN9z2gIguqxDJmz6LBw9h
PYHKN-VDx0G8UgPtF-Dls8EiizZi2L4A3WMX-amh0o-5hd5YHP5qt2petUlwwZkI49CQPT41w-D4j7nvzDZ7j0x5YyAiLOIOvD8nMOjTLXn08GMmFgWk05cW
eg2K5O5tMBsomC46HqLz8pSYPsj9bcIz8wCF4qjWgIzqv6CsI5ObAt_xkshW_KVYmQ
```

将命令显示的 token 值复制到网页中，之后登录成功



K8S威胁矩阵

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Images from a private registry	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account		Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files		
Exposed Dashboard	SSH server running inside container				Access managed identity credential	Instance Metadata API	Writable volume mounts on the host		
Exposed sensitive interfaces	Sidecar injection				Malicious admission controller		Access Kubernetes dashboard		
							Access tiller endpoint		
							CoreDNS poisoning		
							ARP poisoning and IP spoofing		

= New technique

= Deprecated technique

初始访问

初始访问策略包括用于获取资源访问权的技术。在容器化环境中，这些技术可以首先访问集群。这种访问可以直接通过集群管理层实现，也可以通过访问部署在集群上的恶意或易受攻击的资源来实现。

- 使用云凭据

如果 Kubernetes 集群部署在公共云中（例如，Azure 中的 AKS、GCP 中的 GKE 或 AWS 中的 EKS），受损的云凭证可能导致集群接管。有权访问云帐户凭据的攻击者可以访问集群的管理层。

- 恶意镜像注入

在集群中运行恶意镜像可能会损害集群。访问私有注册中心的攻击者可以在注册表中植入他们自己的受损镜像。然后后者可以由用户拉取。此外，用户经常使用来自可能是恶意的公共注册中心（例如 Docker Hub）的不受信任的镜像。

基于不受信任的基础镜像构建镜像也会导致类似的结果。

- Kubeconfig 文件

kubectl 也使用 kubeconfig 文件，其中包含有关 Kubernetes 集群的详细信息，包括它们的位置和凭据。如果集群托管为云服务（例如 AKS 或 GKE），则此文件通过云命令下载到客户端（例如，AKS 的“az aks get-credential”或 GKE 的“gcloud container clusters get-credentials”）。

如果攻击者可以访问此文件，例如通过受感染的客户端，他们可以使用它来访问集群。

- 易受攻击的应用程序

在集群中运行面向公众的易受攻击的应用程序可以启用对集群的初始访问。运行易受远程代码执行漏洞 (RCE) 攻击的应用程序的容器可能会被利用。如果将服务帐户挂载到容器（Kubernetes 中的默认行为），攻击者将能够使用此服务帐户凭据向 API 服务器发送请求。

- 暴露的敏感接口

将敏感接口暴露给互联网会带来安全风险。一些流行的框架并不打算暴露在互联网上，因此默认情况下不需要身份验证。因此，将它们暴露在互联网上会允许未经身份验证地访问敏感接口，这可能会导致恶意行为者在集群中运行代码或部署容器。例如 Kubernetes 仪表板。

执行

执行策略包括攻击者用来在集群内运行代码的技术。

- 执行到容器中

拥有权限的攻击者可以使用 exec 命令（“kubectl exec”）在集群中的容器中运行恶意命令。在这种方法中，攻击者可以使用合法的镜像，例如操作系统镜像（例如，Ubuntu）作为后门容器，并通过“kubectl exec”远程运行他们的恶意代码。

- 新容器

攻击者可能会尝试通过部署容器在集群中运行他们的代码。有权在集群中部署 pod**或控制器**（例如 DaemonSet\ReplicaSet\Deployment）的攻击者可以创建新的资源来运行他们的代码。

- 应用程序利用

部署在集群中且容易受到远程代码执行漏洞或最终允许执行代码的漏洞的应用程序，使攻击者能够在集群中运行代码。如果将服务帐户挂载到容器（Kubernetes 中的默认行为），攻击者将能够使用此服务帐户凭据向 API 服务器发送请求。

- 在容器内运行的 SSH 服务器

在容器内运行的 SSH 服务器可能会被攻击者使用。如果攻击者获得容器的有效凭据，无论是通过蛮力尝试还是通过其他方法（例如网络钓鱼），他们都可以使用它通过 SSH 远程访问容器。

- Sidecar 注入

Kubernetes Pod 是一组具有共享存储和网络资源的一个或多个容器。Sidecar 容器是一个术语，用于描述位于主容器旁边的附加容器。例如，服务网格代理在应用程序的 pod 中作为 sidecar 运行。攻击者可以通过向集群中的合法 pod 注入 sidecar 容器来运行他们的代码并隐藏他们的活动，而不是在集群中运行他们自己的独立 pod。

持久性

持久性策略由攻击者用来保持对集群的访问权的技术组成，以防他们失去最初的立足点。

- 后门容器

攻击者在集群中的容器中运行他们的恶意代码。通过使用 Kubernetes 控制器（例如 DaemonSets 或 Deployments），攻击者可以确保在集群中的一个或所有节点中运行恒定数量的容器。

- 可写 hostPath 挂载

hostPath 卷将目录或文件从主机安装到容器。有权在集群中创建新容器的攻击者可能会创建一个具有可写 hostPath 卷的容器，并在底层主机上获得持久性。例如，后者可以通过在主机上创建一个 cron 作业来实现。

- Kubernetes 定时任务

Kubernetes Job 是一个控制器，它创建一个或多个 Pod，并确保指定数量的 Pod 成功终止。Kubernetes Job 可用于运行为批处理作业执行有限任务的容器。Kubernetes CronJob 用于调度 Jobs。攻击者可能会使用 Kubernetes CronJob 来安排恶意代码的执行，这些代码将作为集群中的容器运行。

- 恶意准入控制器

准入控制器是一个 Kubernetes 组件，它拦截并可能修改对 Kubernetes API 服务器的请求。有两种类型的准入控制器：验证控制器和变异控制器。顾名思义，变异准入控制器可以**修改拦截的请求并更改其属性**。*Kubernetes 有一个名为MutatingAdmissionWebhook*的内置通用准入控制器。此准入控制器的行为由用户在集群中部署的准入 webhook 确定。攻击者可以使用此类 webhook 来获得集群中的持久性。例如，攻击者可以拦截和修改集群中的 Pod 创建操作，并将其恶意容器添加到每个创建的 Pod 中。

权限提升

权限提升策略包括攻击者用来在环境中获得比他们当前拥有的权限更高的权限的技术。在容器化环境中，这可能包括从容器访问节点、在集群中获得更高的权限，甚至访问云资源。

- 特权容器

特权容器是具有主机所有功能的容器，它消除了常规容器的所有限制。实际上，这意味着特权容器几乎可以直接在主机上执行的所有操作。获得对特权容器的访问权限或有权创建新的特权容器（例如，通过使用受感染 pod 的服务帐户）的攻击者可以访问主机的资源。

- 集群管理员绑定

基于角色的访问控制 (RBAC) 是 Kubernetes 中的一项关键安全功能。RBAC 可以限制集群中各种身份的允许操作。Cluster-admin 是 Kubernetes 中内置的高权限角色。有权在集群中创建绑定和集群绑定的攻击者可以创建与集群管理员 ClusterRole 或其他高权限角色的绑定。

- 主机路径挂载

攻击者可以使用 hostPath 挂载来访问底层主机，从而从容器访问到主机。

- 访问云资源

如果 Kubernetes 集群部署在云中，在某些情况下，攻击者可以利用他们对单个容器的访问权来访问集群外的其他云资源。例如，在 AKS 中，每个节点都包含存储在**/etc/kubernetes/azure.json**中的服务主体凭据。AKS 使用此服务主体来创建和管理群集操作所需的 Azure 资源。

默认情况下，服务主体在集群的资源组中具有参与者权限。访问此服务主体文件（例如，通过 hostPath 挂载）的攻击者可以使用其凭据来访问或修改云资源。

防御规避

防御规避策略包括攻击者用来避免检测和隐藏其活动的技术。

- 清除容器日志

攻击者可能会删除受感染容器上的应用程序或操作系统日志，以试图阻止检测到他们的活动。

- 删除 Kubernetes 事件

Kubernetes 事件是记录集群中资源状态更改和故障的 Kubernetes 对象。示例事件是容器创建、图像拉取或节点上的 pod 调度。

Kubernetes 事件对于识别集群中发生的更改非常有用。因此，攻击者可能想要删除这些事件（例如，通过使用：“kubectl delete events- all”）以避免在集群中检测到他们的活动。

- Pod/容器名称相似度

由 Deployment 或 DaemonSet 等控制器创建的 Pod 在其名称中具有随机后缀。攻击者可以利用这一事实并命名他们的后门 pod，因为它们是由现有控制器创建的。例如，攻击者可以创建一个名为*coredns-{random suffix}*的恶意 pod ，它看起来与 CoreDNS 部署相关。

此外，攻击者可以将他们的容器部署在管理容器所在的 kube-system 命名空间中。

- 从代理服务器连接

攻击者可能会使用代理服务器来隐藏其原始 IP。具体来说，攻击者经常使用匿名网络（例如 TOR）进行活动。这可用于与应用程序本身或 API 服务器进行通信。

凭证访问

凭据访问策略包括攻击者用来窃取凭据的技术。

在容器化环境中，这包括正在运行的应用程序的凭证、身份、存储在集群中的机密或云凭证。

- 列出 Kubernetes 机密

Kubernetes 机密是一个对象，它允许用户存储和管理敏感信息，例如集群中的密码和连接字符串。可以在 pod 配置中通过引用来使用 Secret。有权从 API 服务器检索机密的攻击者（例如，通过使用 pod 服务帐户）可以访问敏感信息，其中可能包括各种服务的凭据。

- 挂载服务主体

当集群部署在云中时，在某些情况下，攻击者可以利用他们对集群中容器的访问来获取云凭证。例如，在 AKS 中，每个节点都包含服务主体凭据。（详见“4：访问云资源”。）

- 访问容器服务帐号

服务帐户 (SA) 代表 Kubernetes 中的应用程序身份。默认情况下，SA 会挂载到集群中创建的每个 pod。使用 SA，pod 中的容器可以向 Kubernetes API 服务器发送请求。访问 Pod 的攻击者可以访问 SA 令牌（位于*/var/run/secrets/kubernetes.io/serviceaccount/token*）并根据 SA 权限在集群中执行操作。如果未启用 RBAC，则 SA 在集群中具有无限权限。如果启用了 RBAC，则其权限由与其关联的 RoleBindings \ ClusterRoleBindings 确定。

- 配置文件中的应用程序凭据

开发人员将机密存储在 Kubernetes 配置文件中，例如 pod 配置中的环境变量。此类行为在 Azure 安全中心监控的群集中很常见。有权访问这些配置的攻击者，通过查询 API 服务器或访问开发人员端点上的这些文件，可以窃取存储的秘密并使用它们。

- 访问托管身份凭据

托管身份是由云提供商管理的身份，可以分配给云资源，例如虚拟机。这些身份用于向云服务进行身份验证。身份的秘密完全由云提供商管理，无需管理凭据。应用程序可以通过访问实例元数据服务 (IMDS) 来获取身份的令牌。访问 Kubernetes pod 的攻击者可以利用他们对 IMDS 端点的访问权来获取托管身份的令牌。使用令牌，攻击者可以访问云资源。

- 恶意准入控制器

除了持久性之外，恶意准入控制器还可用于访问凭据。Kubernetes 中的内置准入控制器之一是*ValidatingAdmissionWebhook*。与*MutatingAdmissionWebhook*一样，此准入控制器也是通用的，其行为由部署在集群中的准入 webhook 决定。攻击者可以使用此 Webhook 拦截对 API 服务器的请求、记录机密和其他敏感信息。

发现

发现策略包括攻击者用来探索他们获得访问权限的环境的技术。这种探索有助于攻击者执行横向移动并获得对额外资源的访问权限。

- 访问 Kubernetes API 服务器

Kubernetes API 服务器是集群的网关。通过向 RESTful API 发送各种请求来执行集群中的操作。集群的状态，包括部署在上面的所有组件，可以由 API 服务器检索。攻击者可能会发送 API 请求来探测集群并获取有关集群中容器、机密和其他资源的信息。

- 访问 Kubelet API

Kubelet 是安装在每个节点上的 Kubernetes 代理。Kubelet 负责正确执行分配给节点的 pod。Kubelet 公开了一个不需要身份验证的只读 API 服务（TCP 端口 10255）。对主机具有网络访问权限的攻击者（例如，通过在受感染的容器上运行代码）可以向 Kubelet API 发送 API 请求。具体查询*https://[NODE IP]:10255/pods/*会检索节点上正在运行的 pod。*https://[NODE IP]:10255/spec/*检索有关节点本身的信息，例如 CPU 和内存消耗。

- 网络映射

攻击者可能会尝试映射集群网络以获取有关正在运行的应用程序的信息，包括扫描已知漏洞。默认情况下，Kubernetes 中对 Pod 通信没有限制。因此，获得单个容器访问权限的攻击者可能会使用它来探测网络。

- 访问 Kubernetes 仪表板

Kubernetes 仪表板是一个基于 Web 的 UI，用于监控和管理 Kubernetes 集群。仪表板允许用户使用其服务帐户（kubernetes-dashboard）在集群中执行操作，权限由该服务帐户的绑定或集群绑定确定。获得集群中容器访问权限的攻击者可以使用其对仪表板 pod 的网络访问。因此，攻击者可以使用仪表板的身份检索有关集群中各种资源的信息。

- 实例元数据 API

云提供商提供实例元数据服务，用于检索有关虚拟机的信息，例如网络配置、磁盘和 SSH 公钥。虚拟机可以通过只能从虚拟机内部访问的不可路由的 IP 地址访问此服务。获得容器访问权限的攻击者可能会查询元数据 API 服务以获取有关底层节点的信息。

横向移动

横向移动策略包括攻击者用来在受害者环境中移动的技术。在容器化环境中，这包括通过对一个容器的给定访问权获得对集群中各种资源的访问权，从容器获得对底层节点的访问权，或获得对云环境的访问权。

- 访问云资源

攻击者可能会从受感染的容器转移到云环境。

- 容器服务帐号

获得集群中容器访问权限的攻击者可以使用挂载的服务帐户令牌向 API 服务器发送请求，并获得对集群中其他资源的访问权限。

- 集群内部网络

Kubernetes 网络行为允许集群中 pod 之间的流量作为默认行为。获得对单个容器的访问权限的攻击者可以使用它来实现对集群中另一个容器的网络可达性。

- 配置文件中的应用程序凭据

开发人员将机密存储在 Kubernetes 配置文件中，例如，作为 pod 配置中的环境变量。使用这些凭据，攻击者可以访问集群内外的其他资源。

- 主机上挂载可写卷

攻击者可能会尝试从受感染的容器中获取对底层主机的访问权限。

- 访问 Kubernetes 仪表板

有权访问 Kubernetes 仪表板的攻击者可以管理集群资源，还可以使用仪表板的内置“exec”功能在集群中的各种容器上运行他们的代码。

- CoreDNS 中毒

CoreDNS 是用 Go 编写的模块化域名系统 (DNS) 服务器，由云原生计算基金会 (CNCF) 托管。CoreDNS 是 Kubernetes 中使用的主要 DNS 服务。CoreDNS 的配置可以通过一个名为 corefile 的文件进行修改。在 Kubernetes 中，此文件存储在位于 kube-system 命名空间的 ConfigMap 对象中。如果攻击者拥有修改 ConfigMap 的权限，例如使用容器的服务帐户，他们就可以改变集群 DNS 的行为，毒化它，并获取其他服务的网络身份。

- ARP中毒和IP欺骗

Kubernetes 有许多可以在集群中使用的网络插件（容器网络接口或 CNI）。Kubenet 是基本的，在许多情况下是默认的网络插件。在此配置中，在每个节点 (cbr0) 上创建一个网桥，各种 pod 使用 veth 对连接到该网桥。跨 Pod 流量通过一个 2 级组件的网桥这一事实意味着在集群中执行 ARP 中毒是可能的。因此，如果攻击者访问了集群中的某个 pod，就可以进行 ARP 毒化，并欺骗其他 pod 的流量。通过使用这种技术，攻击者可以在网络级别执行多种攻击，从而导致横向移动，例如 DNS 欺骗或窃取其他 pod 的云身份。

影响

影响策略包括攻击者用来破坏、滥用或破坏环境正常行为的技术。

- 数据销毁

攻击者可能会尝试破坏集群中的数据和资源。这包括删除部署、配置、存储和计算资源。

- 资源劫持

攻击者可能滥用受损资源来运行任务。一种常见的滥用行为是使用受损资源来运行数字货币挖掘。有权访问集群中的容器或有权创建新容器的攻击者可以将它们用于此类活动。

- 拒绝服务

攻击者可能会尝试执行拒绝服务攻击，从而使合法用户无法使用该服务。在容器集群中，这包括尝试阻止容器本身、底层节点或 API 服务器的可用性。

采集(Collection)

在 Kubernetes 中，采集由攻击者用来从集群或通过使用集群收集数据的技术组成。

- 来自私人注册表的图像

集群中运行的镜像可以存储在私有注册中心中。为了拉取这些镜像，容器运行时引擎（例如 Docker 或 containerd）需要具有这些注册中心的有效凭据。

如果注册中心由云托管商提供，则在 Azure Container Registry (ACR) 或 Amazon Elastic Container Registry (ECR) 等服务中，云凭证用于对注册中心进行身份验证。

如果攻击者获得对集群的访问权，在某些情况下，**他们可以获得对私有注册中心的访问权并拉取其镜像**。例如，攻击者可以使用“访问托管身份凭证”技术中描述的托管身份令牌。同样，在 EKS 中，攻击者可以使用默认绑定到节点的 IAM 角色的 AmazonEC2ContainerRegistryReadOnly 策略。

K8S安全加固

Pod安全策略

Kubernetes Pod 安全策略的例子

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames:
'docker/default,runtime/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName: 'runtime/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default'
spec:
  privileged: false # 需要防止升级到 root
    allowPrivilegeEscalation: false
    requiredDropCapabilities:
      - ALL
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    - 'persistentVolumeClaim' # 假设管理员设置的 persistentVolumes 是安全的
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: 'MustRunAsNonRoot' # 要求容器在没有 root 的情况下运行 seLinux
    rule: 'RunAsAny' # 假设节点使用的是 AppArmor 而不是 SELinux
    supplementalGroups:
      rule: 'MustRunAs'
      ranges: # 禁止添加到 root 组
        - min: 1
          max: 65535
    runAsGroup:
      rule: 'MustRunAs'
      ranges: # 禁止添加到 root 组
        - min: 1
          max: 65535
    fsGroup:
      rule: 'MustRunAs'
      ranges: # 禁止添加到 root 组
        - min: 1
          max: 65535
  readOnlyRootFilesystem: true
```

增加审计策略

```yaml
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: RequestResponse
# 这个审计策略记录了 RequestResponse 级别的所有审计事件
```

云原生安全架构

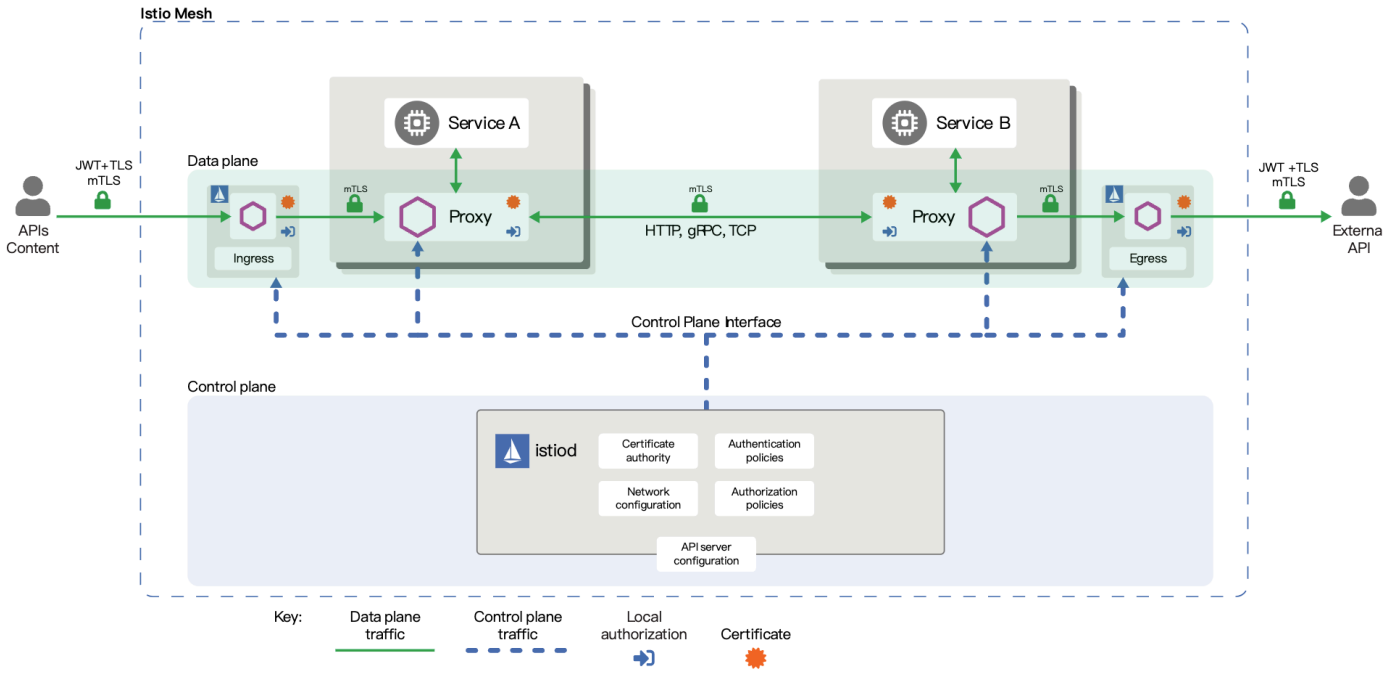
Istio 安全功能提供强大的身份、强大的策略、透明的 TLS 加密以及身份验证、授权和审计 (AAA) 工具来保护您的服务和数据。Istio 安全的目标是：

- 默认安全性：无需更改应用程序代码和基础架构
- 纵深防御：与现有安全系统集成，提供多层防御
- 零信任网络：在不信任网络上构建安全解决方案

istio 是什么

> Istio 提供一种简单的方式来为已部署的服务建立网络，该网络具有负载均衡、服务间认证、监控等功能，而不需要对服务的代码做任何改动。

- istio 适用于容器或虚拟机环境（特别是 k8s），兼容异构架构。
- istio 使用 sidecar（边车模式）代理服务的网络，不需要对业务代码本身做任何的改动。
- HTTP、gRPC、WebSocket 和 TCP 流量的自动负载均衡。
- istio 通过丰富的路由规则、重试、故障转移和故障注入，可以对流量行为进行细粒度控制；支持访问控制、速率限制和配额。
- istio 对出入集群入口和出口中所有流量的自动度量指标、日志记录和跟踪。



Istio 中的安全性涉及多个组件：

- 用于密钥和证书管理的证书颁发机构 (CA)
- 配置 API 服务器分发给代理：
    - 身份验证策略
    - 授权策略
    - 安全命名信息
- Sidecar 和外围代理作为策略执行点 (PEP) 来保护客户端和服务器之间的通信。
- 一组 Envoy 代理扩展来管理遥测和审计