2022 年全国职业院校技能大赛云计算赛项

容器云平台
用户手册 v2.0

2022 年 5 月

目录

1.前言

1.1 编写目的

本文档主要用于指导参赛选手进行容器云平台的部署，并对平台运维提供参考。本文档详细介绍了容器云平台部署的环境要求、过程步骤及基本使用等。

1.2 使用对象

本文档为全国职业院校技能大赛“云计算”赛项容器云平台部署手册，仅适用于参赛选手，使用者需熟悉 Linux 的基础操作。

2.容器云平台部署

2.1 运行环境

2.1.1 软件版本

底层基础平台部署所涉及的软件及版本如下：

操作系统版本：CentOS_7.9.2009

Docker 版本：20.10.12

Kubernetes 版本：1.22.1

KubeVirt 版本：0.47.1

Harbor 版本：2.3.4

Istio 版本：1.12.0

2.1.4 节点规划

集群节点数要求不低于两台。节点规划如下：

节点 IP	角色	备注
10.24.2.10	Master	Kubernetes 集群 master 节点、Harbor 仓库节点
10.24.2.11	Worker	Kubernetes 集群 node 节点

为方便部署，建议保持所有节点密码一致，完成后再统一修改节点密码。

2.2 基础环境准备

将提供的安装包 chinaskills_cloud_paas_v2.0.2.iso 上传至 master 节点/root 目录，并解压到/opt 目录：

```
[root@localhost ~]# mount -o loop chinaskills_cloud_paas_v2.0.2.iso /mnt/
[root@localhost ~]# cp -rfv /mnt/* /opt/
[root@localhost ~]# umount /mnt/
```

2.2.1 安装 kubeeasy

kubeeasy 为 Kubernetes 集群专业部署工具，极大的简化了部署流程。其特性如下：

- 全自动化安装流程；
- 支持 DNS 识别集群；
- 支持自我修复：一切都在自动扩缩组中运行；
- 支持多种操作系统（如 Debian、Ubuntu 16.04、CentOS7、RHEL 等）；
- 支持高可用。

在 master 节点安装 kubeeasy 工具：

```
[root@localhost ~]# mv /opt/kubeeasy /usr/bin/kubeeasy
```

2.2.2 安装依赖包

此步骤主要完成 docker-ce、git、unzip、vim、wget 等工具的安装。

在 master 节点执行以下命令完成依赖包的安装：

```
[root@localhost ~]# kubeeasy install depend \
   --host 10.24.2.10,10.24.2.11 \
   --user root \
   --password 000000 \
   --offline-file /opt/dependencies/base-rpms.tar.gz
```

参数解释如下：

--host：所有主机节点 IP，如：10.24.1.2-10.24.1.10，中间用“-”隔开，表示 10.24.1.2 到 10.24.1.10 范围内的所有 IP。若 IP 地址不连续，则列出所有节点 IP，用逗号隔开，如：10.24.1.2,10.24.1.7,10.24.1.9。

--user：主机登录用户，默认为 root。

--password：主机登录密码，所有节点需保持密码一致。

--offline-file：离线安装包路径。

可通过命令“tail -f /var/log/kubeinstall.log”查看安装详情或排查错误。

2.2.3 配置 SSH 免密钥

安装 Kubernetes 集群的时候，需要配置 Kubernetes 集群各节点间的免密登录，方便传

输文件和通讯。

在 master 节点执行以下命令完成集群节点的连通性检测：

```
[root@localhost ~]# kubeeasy check ssh \
   --host 10.24.2.10,10.24.2.11 \
   --user root \
   --password 000000
```

在 master 节点执行以下命令完成集群所有节点间的免密钥配置：

```
[root@localhost ~]# kubeeasy create ssh-keygen \
   --master 10.24.2.10 \
   --worker 10.24.2.11 \
   --user root --password 000000
```

--mater 参数后跟 master 节点 IP，--worker 参数后跟所有 worker 节点 IP。

2.3 部署 Kubernetes 集群

2.3.1 Kubernetes 1.22 简介

2021 年 8 月 4 日，Kubernetes v1.22 正式发布，这是 Kubernetes 在 2021 年发布的第二个版本。此版本包含 53 项增强功能：其中 13 项功能已升级至稳定版，24 项功能顺利步入 beta 阶段，16 项功能刚刚开始 alpha 阶段。另有 3 项功能被彻底弃用。

Kubernetes 的发布周期已经正式由每年 4 次调整为每年 3 次,而 1.22 版本正是调整之后的首个长周期发布版本。随着 Kubernetes 的逐渐成熟，每个发布周期中包含的增强功能数量一直在持续增加。

Kubernetes v1.22 主要特性如下：

（1）Server-side Apply 迎来 GA 通用版本

Server-side Apply 是一种面向 Kubernetes API 服务器的全新字段所有权及对象合并算法。Server-side Apply 通过声明式配置帮助用户及控制器管理其资源，包括以声明方式创建及/或修改对象、发送明确指定的意图等等。经过数个版本的测试之后，Server-side Apply 现已正式进入 GA 通用版阶段。

（2）外部凭证提供程序迎来稳定版

对 Kubernetes 客户端支持凭据插件 sample-exec-plugin 自 1.11 以来，一直处于测试阶段，

随着 Kubernetes 1.22 的发布，现在逐渐稳定。GA 功能集包括对提供交互式登录流程的插件的改进支持，以及许多错误修复。

（3）etcd 升级到 3.5.0

Kubernetes 的默认后端存储 etcd 升级到了新版本：3.5.0。新版本改进了安全性、性能、监控和开发人员体验。 有许多错误修复和一些关键的新功能，例如迁移到结构化日志记录和内置日志轮换。该版本附带了详细的未来路线图，以实施交通过载的解决方案。

（4）内存资源的服务质量

最初，Kubernetes 使用 v1 cgroups API。 使用这种设计，QoS 等级为 Pod 仅适用于 CPU 资源（例如 cpu_shares）。Kubernetes v1.22 现在提供 alpha 版本的 cgroups v2 API 来控制内存分配和隔离。该功能旨在在内存资源争用时提高工作负载和节点可用性，并提高容器生命周期的可预测性。

（5）节点系统交换支持

每个系统管理员或 Kubernetes 用户在设置和使用 Kubernetes 方面都禁用交换空间。在 Kubernetes 1.22 中，现在可以运行 alpha 版本的交换内存的节点。该功能允许管理员选择在 Linux 节点上配置交换，将块存储的一部分视为额外的虚拟内存。

（6）Windows 增强功能和功能

SIG Windows 继续支持不断增长的开发人员社区，发布了开发环境。这些新工具支持多个 CNI 提供程序并且可以在多个平台上运行。通过编译 Windows kubelet 和 kube-proxy，然后将它们与其他 Kubernetes 组件的日常构建一起使用，还有一种从头开始运行 Windows 功能的新方法。

CSI 对 Windows 节点的支持在 1.22 版本中移至 GA 版本中。 在 Kubernetes v1.22 中，Windows 特权容器还是 alpha 功能。为了允许在 Windows 节点上使用 CSI 存储，CSIProxy 为非允许将 CSI 节点插件部署特权 Pod，使用代理在节点上执行特权存储操作。

（7）seccomp 的默认配置文件

kubelet 中添加了默认 seccomp 配置文件，以及新的命令行标志和配置。使用时，新功能提供集群范围的 seccomp 默认值，使用 RuntimeDefaultseccomp 配置文件而不是 Unconfined 默认情况下。这增强了 Kubernetes 部署的默认安全性。安全管理员现在知道工作负载在默认情况下更安全， seccomp 的默认配置为 alpha 功能

（8）使用 kubeadm 实现更安全的控制平面

一个新的 alpha 功能为允许运行 kubeadm 控制平面组件作为非 root 用户。这是一项长期

要求的安全措施 kubeadm. 要尝试它，必须启用 kubeadm 特定的 RootlessControlPlane 功能门。使用此 alpha 功能部署集群时，控制平面以较低的权限运行。

为了 kubeadm，Kubernetes 1.22 还带来了新的 v1beta3 配置 API。改迭代添加了一些长期请求的功能并弃用了一些现有功能。v1beta3 版本现在是首选的 API 版本；v1beta2 API 也仍然可用。

2.3.2 安装 Kubernetes 集群

本次安装的 Kubernetes 版本为 v1.22.1。

在 master 节点执行以下命令部署 Kubernetes 集群：

```
[root@localhost ~]# kubeeasy install kubernetes \
    --master 10.24.2.10 \
    --worker 10.24.2.11 \
    --user root \
    --password 000000 \
    --version 1.22.1 \
    --offline-file /opt/kubernetes.tar.gz
```

部分参数解释如下：

```
--master：Master 节点 IP。
--worker：Node 节点 IP，如有多个 Node 节点用逗号隔开。
--version：Kubernetes 版本，此处只能为 1.22.1。
```

可通过命令“tail -f /var/log/kubeinstall.log”查看安装详情或排查错误。

部署完成后查看集群状态：

```
[root@k8s-master-node1 ~]# kubectl cluster-info
Kubernetes control plane is running at https://apiserver.cluster.local:6443
CoreDNS is running at
https://apiserver.cluster.local:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

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

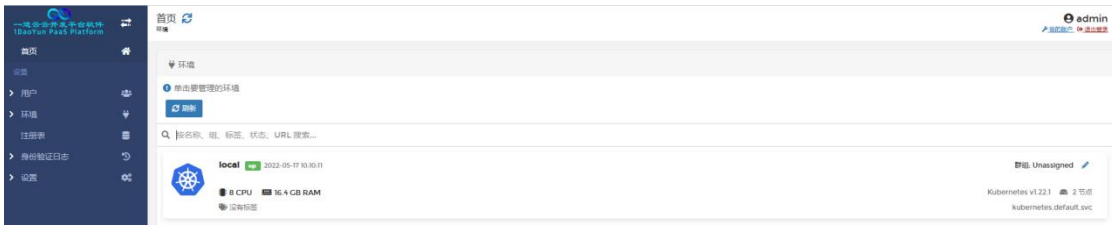
查看节点负载情况：

```
[root@k8s-master-node1 ~]# kubectl top nodes --use-protocol-buffers
NAME                CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k8s-master-node1    389m         4%     6926Mi          43%
k8s-worker-node1    875m         10%    3365Mi          21%
```

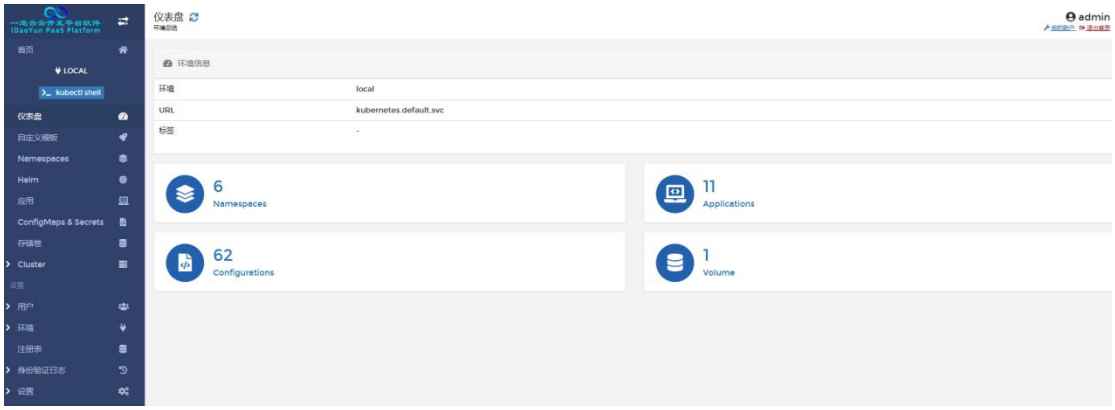
2.3.3 登录一道云云开发平台

在浏览器上访问一道云云开发平台（http://master_IP:30080），如图所示：

设置 admin 用户的密码（000000000000），并登录平台，如图所示：



点击集群名称查看集群概览，如图所示：



2.4 部署 KubeVirt 集群

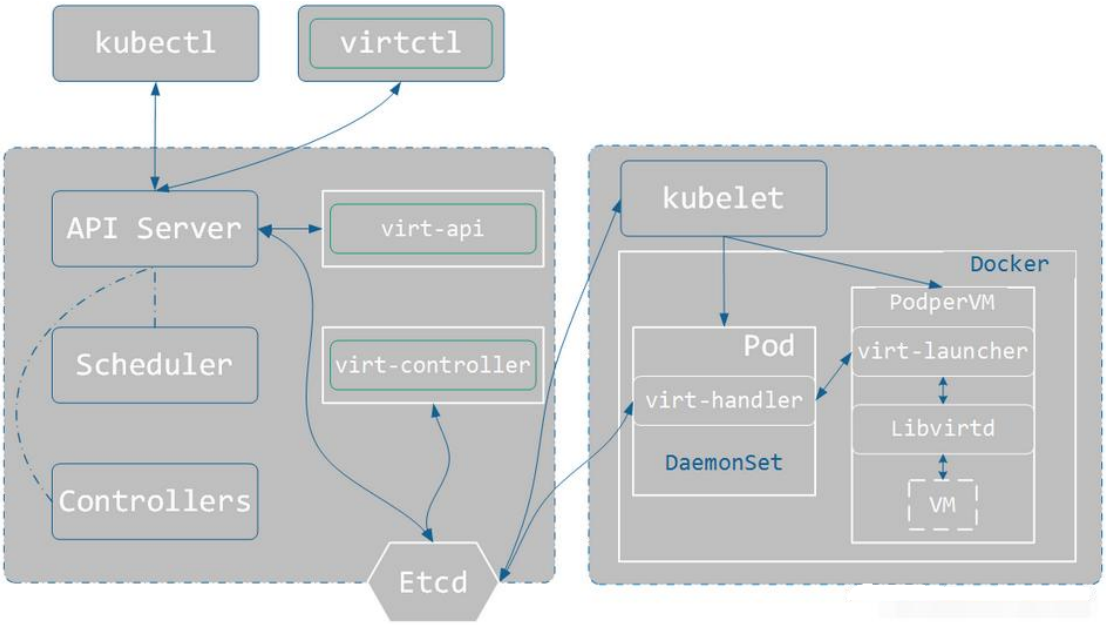
2.4.1 KubeVirt 简介

根据 Garnter 的最新预测，到 2022 年将会有 75%的生产应用全部跑在容器环境之上。基于这个预测，其实至少还有 25%的架构由于技术原因或者是认为原因都将仍然跑在旧的架构之上，这其中虚拟机又会占据其中的大部分份额。所以在容器技术尤其是 Kubernetes 诞生之初，就已经有开源的社区在为如何使用 Kubernetes 纳管虚拟机作为一个重要的功能在开发和贡献，KubeVirt 就是其中之一。

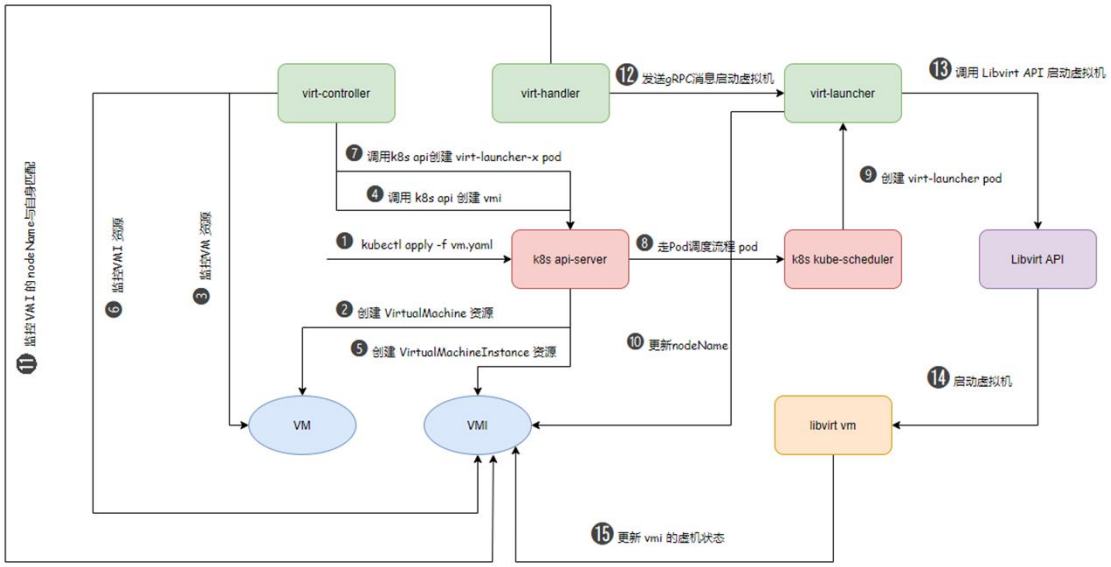
使用 KubeVirt 主要解决了以下两个问题：

- 从技术层面，完全的虚拟机纳管，可以完美迁移因为内核版本过于陈旧以及语言问题而无法迁移到容器的部分应用；
- 从管理和运维层面，符合传统的运维工作方式，以前的 SSH 等运维方式可以完美复用。

KubeVirt 架构如下图所示：



KubeVirt 创建虚拟机的流程如下：



2.4.2 安装 KubeVirt

本次安装的 KubeVirt 版本为 v0.47.1。

在 master 节点执行以下命令安装 KubeVirt：

```
[root@k8s-master-node1 ~]# kubeeasy add --virt kubevirt
```

查看 Pod：

```
[root@k8s-master-node1 ~]# kubectl -n kubevirt get pods
NAME                              READY   STATUS    RESTARTS   AGE
```

virt-api-8d998846b-2xx5m	1/1	Running	0	86s
virt-api-8d998846b-tqwhq	1/1	Running	0	86s
virt-controller-76b86f6965-gz8m4	1/1	Running	0	62s
virt-controller-76b86f6965-pjjv5	1/1	Running	0	62s
virt-handler-hvf45	1/1	Running	0	62s
virt-handler-x7bvj	1/1	Running	0	62s
virt-operator-579f86869c-k9nw4	1/1	Running	0	2m22s
virt-operator-579f86869c-vtrkn	1/1	Running	0	2m22s

2.5 部署 Istio

2.4.1 Istio 简介

上云给 DevOps 团队带来了不小的压力。为了可移植性，开发人员必须使用微服务来构建应用，同时运维人员也正在管理着极端庞大的混合云和多云的部署环境。Istio 允许用户连接、保护、控制和观察服务。

从较高的层面来说，Istio 有助于降低这些部署的复杂性，并减轻开发团队的压力。它是一个完全开源的服务网格，作为透明的一层接入到现有的分布式应用程序里。它也是一个平台，拥有可以集成任何日志、遥测和策略系统的 API 接口。Istio 多样化的特性使您能够成功且高效地运行分布式微服务架构，并提供保护、连接和监控微服务的统一方法。Istio 解决了开发人员和运维人员所面临的从单体应用向分布式微服务架构转变的挑战。

服务网格是用来描述组成这些应用程序的微服务网络以及它们之间的交互。随着服务网格的规模和复杂性不断的增长，它将会变得越来越难以理解和管理。它的需求包括服务发现、负载均衡、故障恢复、度量和监控等。服务网格通常还有更复杂的运维需求，比如 A/B 测试、金丝雀发布、速率限制、访问控制和端到端认证。

Istio 提供了对整个服务网格的行为洞察和操作控制的能力，以及一个完整的满足微服务应用各种需求的解决方案

通过负载均衡、服务间的身份验证、监控等方法，Istio 可以轻松地创建一个已经部署了服务的网络，而服务的代码只需很少更改甚至无需更改。通过在整个环境中部署一个特殊的 sidecar 代理为服务添加 Istio 的支持，而代理会拦截微服务之间的所有网络通信，然后使用其控制平面的功能来配置和管理 Istio，这包括：

- 为 HTTP、gRPC、WebSocket 和 TCP 流量自动负载均衡。
- 通过丰富的路由规则、重试、故障转移和故障注入对流量行为进行细粒度控制。
- 可插拔的策略层和配置 API，支持访问控制、速率限制和配额。
- 集群内（包括集群的入口和出口）所有流量的自动化度量、日志记录和追踪。
- 在具有强大的基于身份验证和授权的集群中实现安全的服务间通信。

Istio 为可扩展性而设计，可以满足不同的部署需求。

2.4.2 安装 Istio

本次安装的 Istio 版本为 v1.12.0。

在 master 节点执行以下命令进行 Istio 服务网格环境的安装：

```
[root@k8s-master-node1 ~]# kubeeasy add --istio istio
```

查看 Pod：

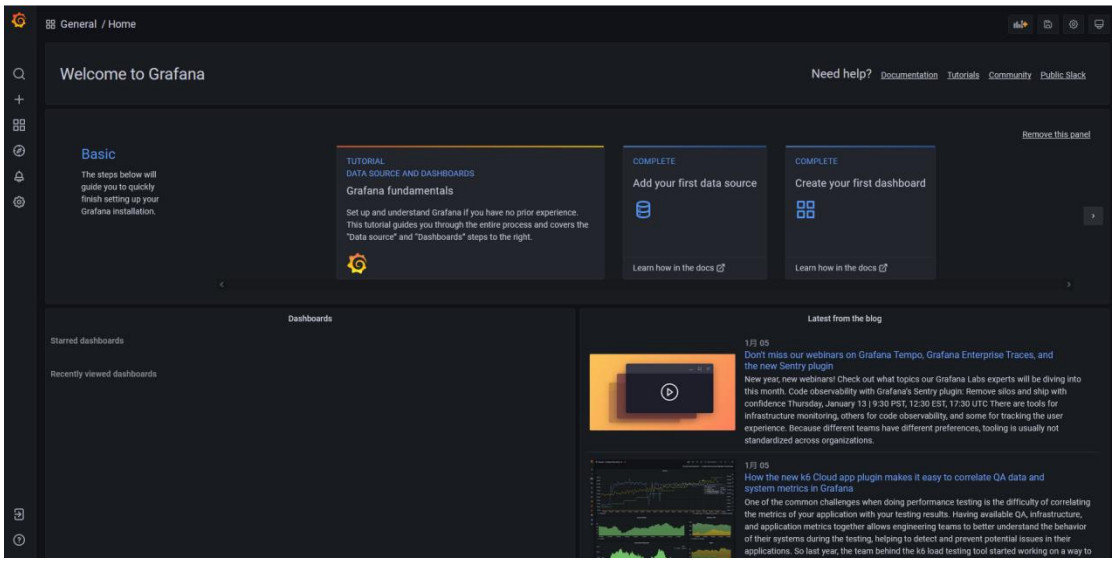
```
[root@k8s-master-node1 ~]# kubectl -n istio-system get pods
NAME                                READY   STATUS    RESTARTS   AGE
grafana-6ccd56f4b6-twwjv            1/1     Running   0          5m15s
istio-egressgateway-7f4864f59c-nxz2l 1/1     Running   0          5m34s
istio-ingressgateway-55d9fb9f-jzhnb 1/1     Running   0          5m34s
istiod-555d47cb65-jwkgp             1/1     Running   0          5m40s
jaeger-5d44bc5c5d-h9t29             1/1     Running   0          5m15s
kiali-79b86ff5bc-v9sfk              1/1     Running   0          5m15s
prometheus-64fd8ccd65-5px64         2/2     Running   0          5m15s
```

查看 Istio 版本信息：

```
[root@k8s-master-node1 ~]# istioctl version
client version: 1.12.0
control plane version: 1.12.0
data plane version: 1.12.0 (2 proxies)
```

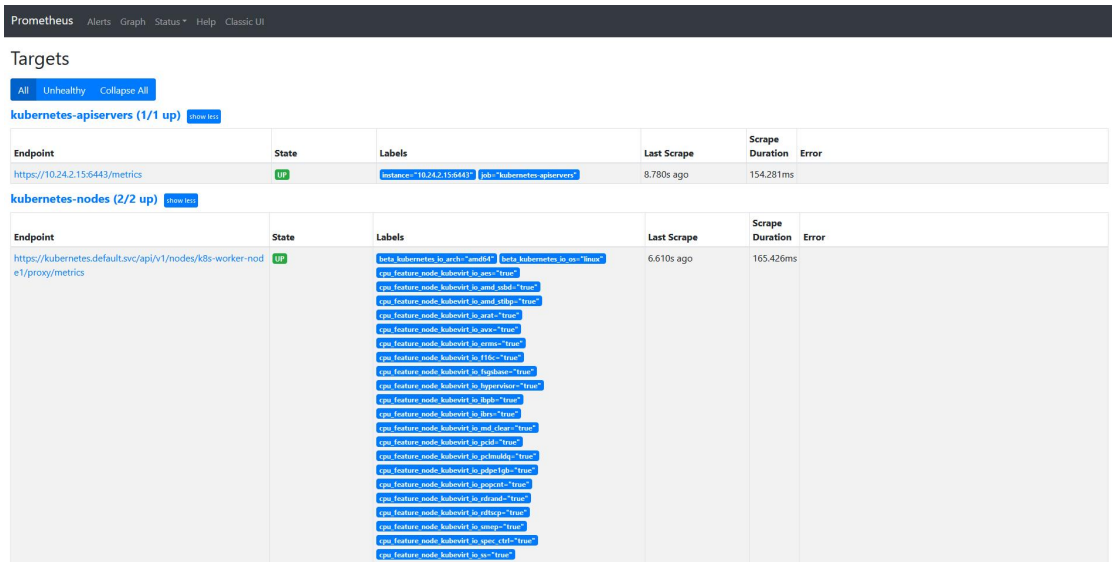
2.4.3 Istio 可视化

访问 Grafana（http://master_IP:33000），如图所示：

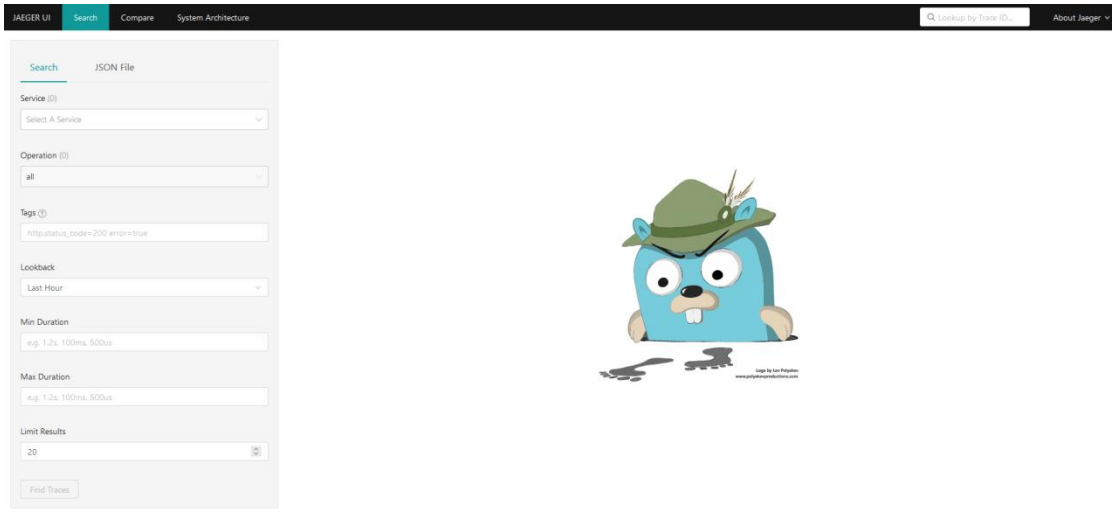


访问 Prometheus（http://master_IP:30090），如图所示：

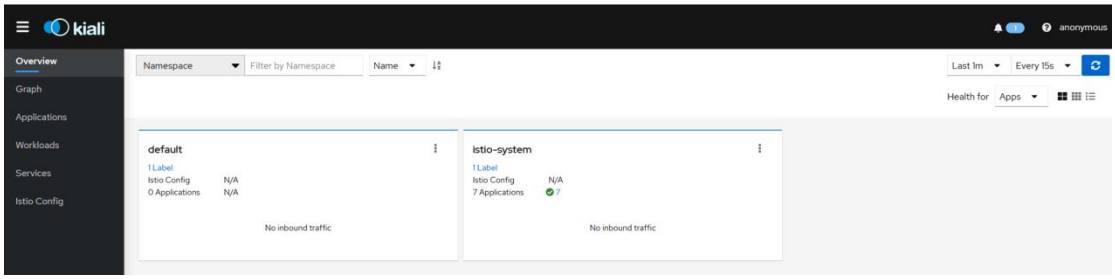
查看 Targets，如图所示：



访问 Jaeger（http://master_IP:30686），如图所示：



访问 Kiali（http://master_IP:20001），如图所示：



2.6 部署 Harbor 仓库

2.6.1 Harbor 仓库简介

Harbor 是一个用于存储和分发 Docker 镜像的企业级 Registry 服务器，通过添加一些企业必需的功能特性，例如安全、标识和管理等，扩展了开源 Docker Distribution。作为一个企业级私有 Registry 服务器，Harbor 提供了更好的性能和安全。提升用户使用 Registry 构建和运行环境传输镜像的效率。Harbor 支持安装在多个 Registry 节点的镜像资源复制，镜像全部保存在私有 Registry 中， 确保数据和知识产权在公司内部网络中管控。另外，Harbor 也提供了高级的安全特性，诸如用户管理，访问控制和活动审计等。

2.6.2 安装 Harbor 仓库

本次安装的 KubeVirt 版本为 2.3.4。

在 master 节点执行以下命令进行 Harbor 仓库的安装：

```
[root@k8s-master-node1 ~]# kubeeasy add --registry harbor
```

部署完成后查看 Harbor 仓库状态：

```
[root@k8s-master-node1 ~]# systemctl status harbor
● harbor.service - Harbor
   Loaded: loaded (/usr/lib/systemd/system/harbor.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2021-12-06 14:34:49 CST; 30s ago
     Docs: http://github.com/vmware/harbor
 Main PID: 7819 (docker-compose)
    Tasks: 12
   Memory: 11.5M
   CGroup: /system.slice/harbor.service
           └─7819 /usr/local/bin/docker-compose -f /opt/harbor/docker-compose.yml up
Hint: Some lines were ellipsized, use -l to show in full.
```

在 Web 端通过 http://master_ip 访问 Harbor，如图所示：

使用管理员账号（admin/Harbor12345）登录 Harbor，如图所示：

2.7 基础运维

2.7.1 重置集群

若集群部署失败或出现故障，可重置集群重新部署，重置命令如下：

```
[root@k8s-master-node1 ~]# kubeeasy reset
```

重置完成后再次执行步骤 2.2.2--2.6.2 即可重新部署集群。

2.7.2 添加节点

在 master 节点执行以下命令安装依赖包：

```
[root@k8s-master-node1 ~]# kubeeasy install depend \
    --host 10.24.2.12 \
    --user root \
    --password 000000 \
    --offline-file /opt/dependencies/base-rpms.tar.gz
```

其中 10.24.2.12 为新增节点的 IP 地址。

在 master 节点执行以下命令即可加入集群：

```
[root@k8s-master-node1 ~]# kubeeasy add \
    --worker 10.24.2.12\
    --user root \
    --password 000000 \
    --offline-file /opt/kubernetes.tar.gz
```

3.容器云平台基础使用

3.1 Kubernetes 集群管理

3.1.1 kubectl 常用命令

创建资源对象：

```
# kubectl create -f xxx.yaml(文件)
# kubectl create -f <directory>(目录下所有文件)
```

查看资源对象：

```
# kubectl get nodes
# kubectl get pods -n <namespace> -o wide
```

描述资源对象：

```
# kubectl describe nodes <node-name>
# kubectl describe pod <pod-name> -n <namespace>
```

删除资源对象：

```
# kubectl delete -f <filename>
# kubectl delete pods,services -l name=<label-name>
# kubectl delete pods --all
```

执行容器的命令：

```
# kubectl exec <pod-name> date(默认使用第一个容器执行 Pod 的 date 命令)
# kubectl exec <pod-name> -c <container-name> date（指定 Pod 中的某个容器执行 date 命令）
# kubectl exec -it <pod-name> -c <container-name> /bin/bash （相当与 docker exec -it <container-name> /bin/bash）
```

查看容器日志：

```
# kubectl logs <pod-name>
# kubectl logs -f <pod-name> -c <container-name> (相当于 tail -f 命令)
```

3.1.2 kubectl 格式化输出

显示 Pod 的更多信息：

```
# kubectl get pods -n <namespace> -o wide
```

以 yaml 格式显示：

```
# kubectl get pods -n <namespace> -o yaml
```

以自定义列显示 Pod 信息：

```
# kubectl get pod <pod-name> -n <namespace> -o
custom-columns=NAME:.metadata.name,"ANNOTATIONS":.metadata.annotations
```

基于文件的自定义列名输出：

```
# kubectl get pods <pod-name> -o=custom-columns-file=template.txt
```

输出结果排序：

```
# kubectl get pods --sort-by=.metadata.name
```

3.2 KubeVirt 集群管理

3.2.1 基本使用

创建 vmi：

```
# kubectl create -f vmi.yaml
```

查看 vmi：

```
# kubectl get vmis
```

删除 vmi：

```
# kubectl delete vmis <vmi-name>
```

3.2.2 virtctl 工具

virtctl 是 KubeVirt 自带的类似于 kubectl 的命令行工具，可以直接管理虚拟机，可以控制虚拟机的 start、stop、restart 等。

启动虚拟机：

```
# virtctl start <vmi-name>
```

停止虚拟机：

```
# virtctl stop <vmi-name>
```

重启虚拟机：

```
# virtctl restart <vmi-name>
```

3.3 Istio 管理

3.3.1 istioctl 基本使用

istioctl 用于在 Istio 系统中创建、列出、修改以及删除配置资源。

可用的路由和流量管理配置类型有：virtualservice、gateway、destinationrule、serviceentry、httpapispec、httpapispecbinding、quotaspec、quotaspecbinding、servicerole、servicerolebinding、policy。

使用下面命令展示 istioctl 可以访问到的 Istio 配置档的名称：

```
# istioctl profile list
Istio configuration profiles:
    default
    demo
    empty
    external
    minimal
    openshift
    preview
    remote
```

展示配置档的配置信息：

```
# istioctl profile dump demo
```

显示配置文件的差异：

```
# istioctl profile diff default demo
```

可以使用 proxy-status 或 ps 命令概览服务网格：

```
# istioctl proxy-status
```

如果输出列表中缺少某个代理则意味着它当前未连接到 Polit 实例，所以它无法接收到任何配置。此外，如果它被标记为 stale，则意味着存在网络问题或者需要扩展 Pilot。

istioctl 允许使用 proxy-config 或者 pc 命令检索代理的配置信息。

检索特定 Pod 中 Envoy 实例的集群配置的信息：

```
# istioctl proxy-config cluster <pod-name> [flags]
```

检索特定 Pod 中 Envoy 实例的 bootstrap 配置的信息：

```
# istioctl proxy-config bootstrap <pod-name> [flags]
```

检索特定 Pod 中 Envoy 实例的监听器配置的信息：

```
# istioctl proxy-config listener <pod-name> [flags]
```

检索特定 Pod 中 Envoy 实例的路由配置的信息：

```
# istioctl proxy-config route <pod-name> [flags]
```

检索特定 Pod 中 Envoy 实例的 endpoint 配置的信息：

```
# istioctl proxy-config endpoints <pod-name> [flags]
```

3.4 Helm 工具

Helm 是 Kubernetes 的包管理器，类似于 Python 的 pip 和 CentOS 的 yum，主要用来管理 Charts。Helm Chart 是用来封装 Kubernetes 原生应用程序的一系列 YAML 文件。可以在部署应用的时候自定义应用程序的一些 Metadata，以便于应用程序的分发。

对于应用发布者而言，可以通过 Helm 打包应用、管理应用依赖关系、管理应用版本并发布应用到软件仓库。

对于使用者而言，使用 Helm 后不用需要编写复杂的应用部署文件，可以以简单的方式在 Kubernetes 上查找、安装、升级、回滚、卸载应用程序

3.4.1 helm 常用命令

查看版本信息：

```
# helm version
```

查看当前安装的 Charts：

```
# helm list
```

查询 Charts：

```
# helm search <chart-name>
```

查看 Charts 状态：

```
# helm status redis
```

删除 Charts：

```
# helm delete --purge <chart-name>
```

创建 Charts：

```
# helm create helm_charts
```

测试 Charts 语法：

```
# helm lint
```

打包 Charts：

```
# cd helm_charts && helm package ./
```

查看生成的 yaml 文件：

```
# helm template helm_charts-xxx.tgz
```