

# OpenFaaS 环境搭建指南

本文简单介绍在 window10 环境下，从零开始搭基于 docker 和 k8s 的 OpenFaaS 环境。首先，需要进行 docker 的安装以及 k8s 集群的搭建，然后进行 OpenFaaS 的配置，配置成功后还将进行基于 Python 的简单测试。

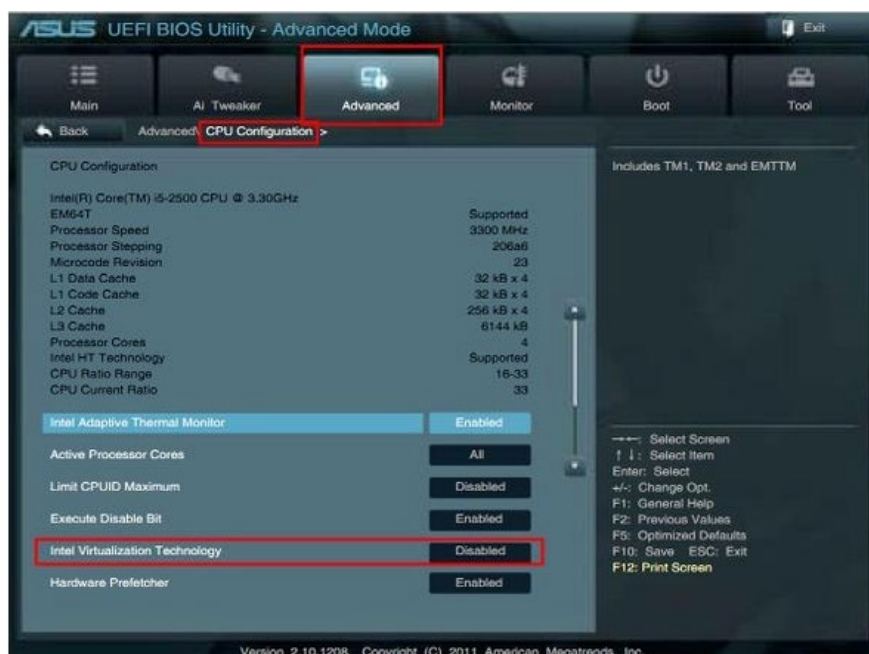
## 1. CPU 虚拟化及 Hyper-V 安装

本小节主要介绍 CPU 虚拟化服务的设置以及 Hyper-V 的常用安装方法。

### 1.1 CPU 虚拟化设置

在 windows 系统中运行 docker 服务需要开启虚拟化服务，否则无法成功安装 docker。为了成功安装 Hyper-V 虚拟化工具，我们首先需要在 BIOS 系统中开启 CPU 虚拟化设置。这里简单以华硕主板进行举例（其它类型主板方式类似）。

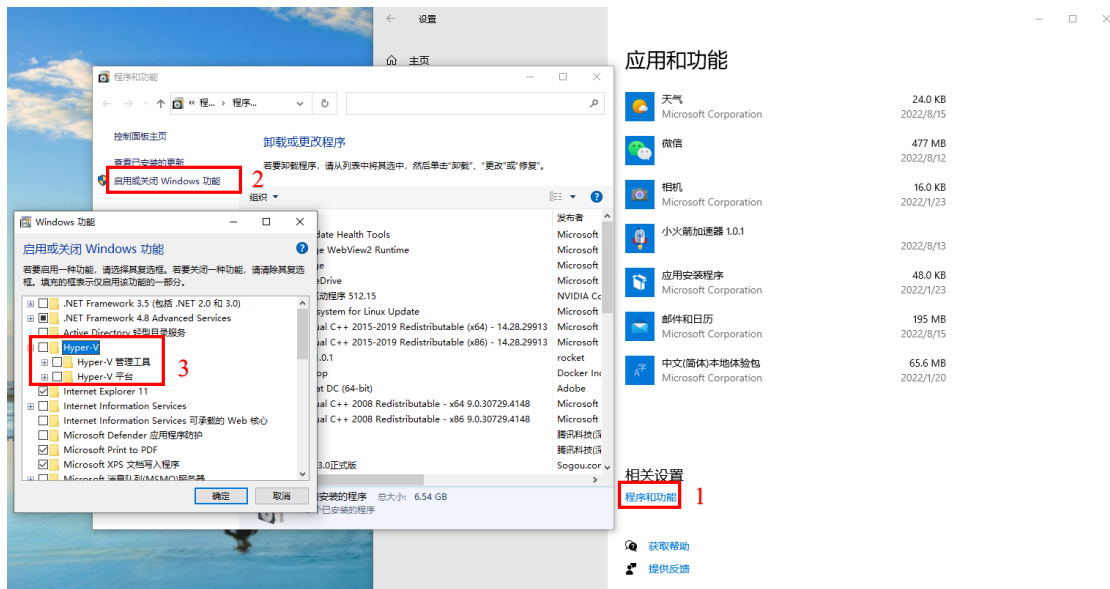
开机时按 F2 或 DEL 键进入 BIOS 设置，紧接着按 F7 进入 Advanced（高级菜单），进入 CPU Configuration。



将 Intel Virtualization Technology (Intel 虚拟化技术) 设置为 Enabled (启用)。如果是 AMD 的 CPU，该选项可能叫做 SVM (Secure VirtualMachine Mode)。最后按 F10 保存设置，重启即可。

## 1.2 Hyper-V 安装

重启电脑后，在按下 win+x 键，打开应用和功能，紧接着点击最下方程序和功能，单机左侧启用或关闭 Windows 功能，找到 Hyper-V 选项，将 Hyper-V 下的 Hyper-V 管理工具和 Hyper-V 平台都勾选，然后点击确定。重启后就完成了 Hyper-V 的安装。



另外，有一种情况 Hyper-V 平台选项为灰色，无法进行勾选。此时可以按下 win+x 键，以管理员方式打开 Windows PowerShell，接着运行以下命令：

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

重启后，按照上面的步骤将 Hyper-V 下的 Hyper-V 管理工具和 Hyper-V 平台都勾选上即可。

## 2. 安装 WSL

WSL 是适用于 Linux 的 Windows 子系统，它随 Windows 系统一起提供，但必须先启用它并安装 Linux 发行版，然后才能开始使用它。

主要参考：<https://docs.microsoft.com/zh-cn/windows/wsl/install>

### 2.1 开始使用

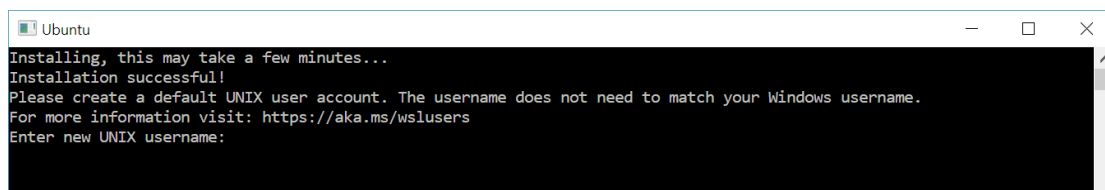
打开 PowerShell（或 Windows 命令提示符）并输入：

---

```
wsl -install
```

## 2.2 设置 Linux 用户名和密码

使用 WSL 安装 Linux 发行版的过程完成后，使用“开始”菜单打开该发行版（默认情况下为 Ubuntu）。系统将要求你为 Linux 发行版创建“用户名”和“密码”。



## 2.3 更新和升级包（可选）

建议使用发行版的首选包管理器定期更新和升级包。对于 Ubuntu 或 Debian，请使用以下命令：

```
sudo apt update && sudo apt upgrade
```

至此，我们就已经成功安装了 WSL，如果想要参考更多有关设置 WSL 开发环境的资料，可以参考：

<https://docs.microsoft.com/zh-cn/windows/wsl/setup/environment>

## 3. Docker 环境

本节主要介绍在 Windows 环境下安装 Docker，前提是必须要完成 Hyper-V 和 WSL 的安装。本章主要参考：<https://zhuanlan.zhihu.com/p/163558476>

### 3.1 下载安装并配置 docker

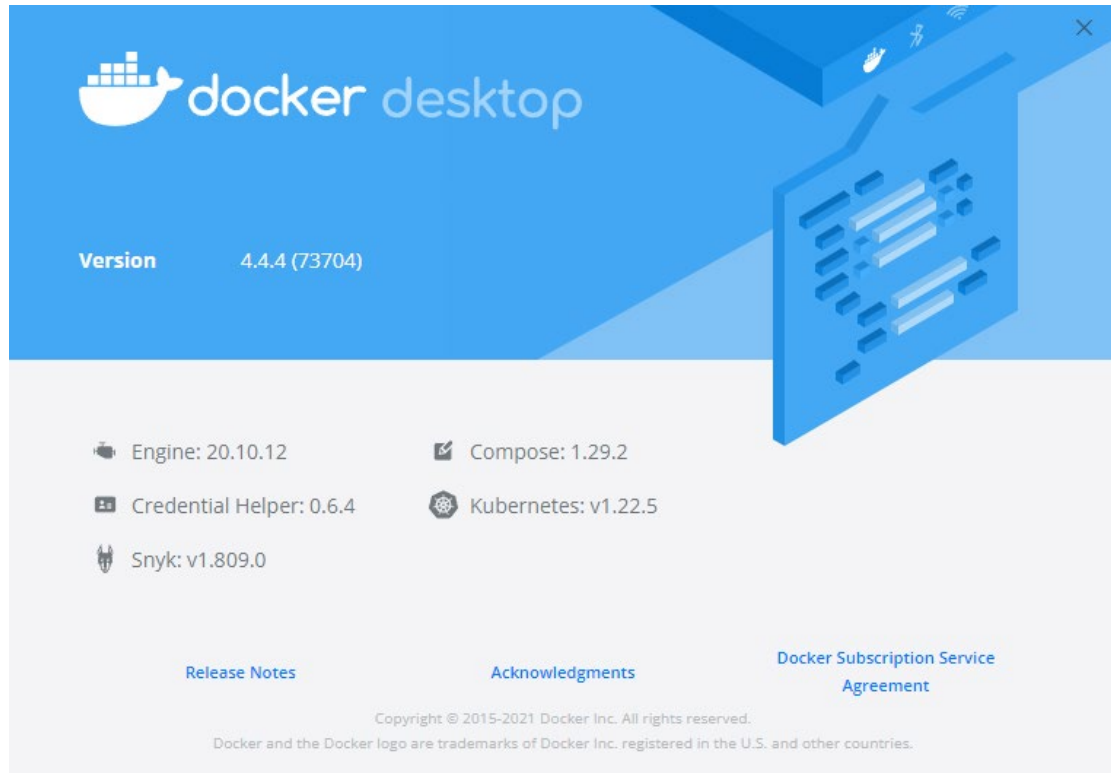
根据调研发现安装 docker 较新版本可能会发生报错，因此我选择 docker 版本为 4.4.4，老版本 docker 可以通过下面的链接进行下载：

<https://docs.docker.com/desktop/windows/release-notes/>

下载完成后直接进行安装，需要勾选的内容保持默认设置即可。安装完成后，一定要右键使用**管理员权限**启动。启动后，先不要开启 k8s，只需要启动 docker

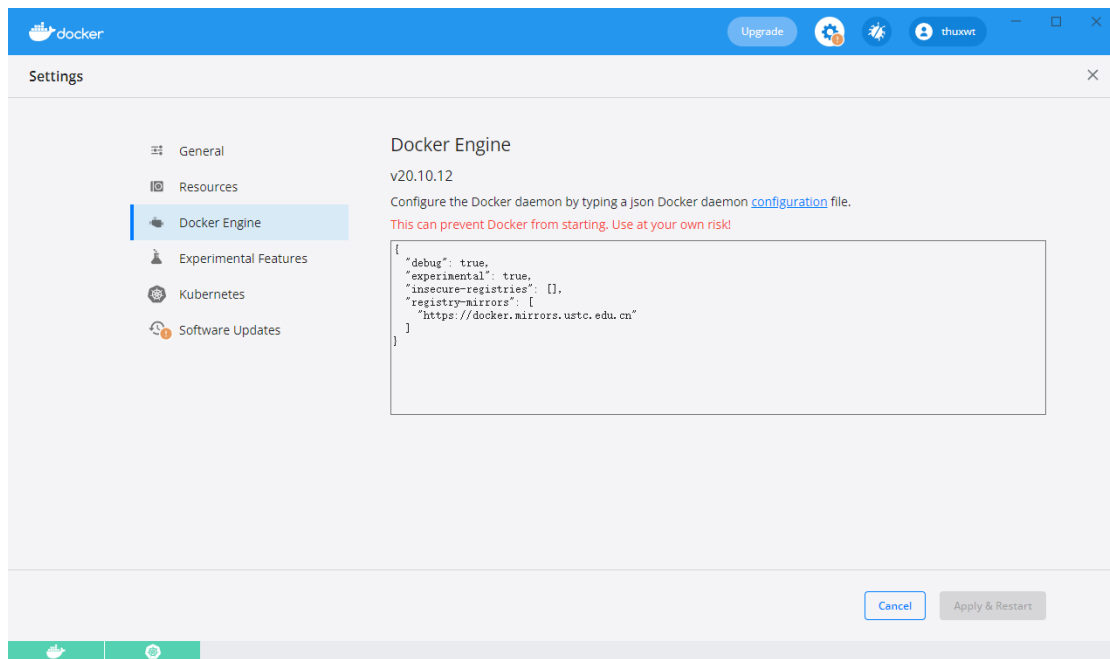
即可。

在右下角 docker 小图标上选择【About Docker Desktop】，会出现以下界面，需要记住自己的 k8s 版本，后面安装会用到。从图中可以看到我的版本是 v1.22.5。



接着，还是在右下角 docker 桌面小图标上右键选择【Dashboard】，点击设置，在左侧点击【Docker Engine】进入下图中的界面，然后填入如下内容，最后点击【Apply & Restart】。

```
{
  "registry-mirrors": [
    "https://docker.mirrors.ustc.edu.cn"
  ],
  "insecure-registries": [],
  "debug": true,
  "experimental": true
}
```



上图中可以发现，我的 k8s 已经处于启动状态，请暂时忽略。

## 3.2 下载必要的镜像

注意，下面的所有操作都要在以管理员权限打开的命令行中执行。

```
# 首先预先从阿里云 Docker 镜像服务下载 k8s 所需要的镜像：
git clone https://github.com/AliyunContainerService/k8s-for-docker-desktop.git

# 切换至对应自己安装的 k8s 版本的分支
git checkout v1.22.5

# 执行下面命令以允许脚本的执行
Set-ExecutionPolicy RemoteSigned

# 然后执行命令开始下载 k8s 所需镜像
.\load_images.ps1
```

这里默认用户已经下载并安装了 git，若没有安装 git 可以参考以下教程进行安装，这里不在赘述：

[https://blog.csdn.net/m0\\_58893670/article/details/121994910](https://blog.csdn.net/m0_58893670/article/details/121994910)

完成上面的操作之后，在命令行输入 `docker images` 可以看到下图的镜像列表中的一部分，只要自己列表中 `k8s.gcr.io` 开头的镜像都存在就可以了。

```
PS C:\Windows\system32> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
thuxwt/statuscode	latest	d0a0908e16ba	2 days ago	94.4MB
thuxwt/add	latest	9b0cfa180e86	2 days ago	87.3MB
ghcr.io/openfaas/queue-worker	0.13.1	eff9b871e09d	11 days ago	9.87MB
ghcr.io/openfaas/faas-netes	0.15.1	57d3bc7e751d	5 weeks ago	74.9MB
ghcr.io/openfaas/gateway	0.23.0	bb2219af32d3	5 weeks ago	30.6MB
ghcr.io/openfaas/basic-auth	0.23.0	ead24e160a59	5 weeks ago	14.2MB
prom/prometheus	v2.36.2	8c01021bb2f4	8 weeks ago	211MB
ghcr.io/openfaas/alpine	latest	35f344be6bb8	5 months ago	14.2MB
ghcr.io/openfaas/nodeinfo	latest	16631c6bf8f3	5 months ago	133MB
k8s.gcr.io/ingress-nginx/controller	v1.1.1	2461b2698ddc	7 months ago	285MB
docker/desktop-kubernetes	kubernetes-v1.22.5-cni-v0.8.5-critools-v1.17.0-debian	88c30feb8fa7	7 months ago	294MB
k8s.gcr.io/kube-apiserver	v1.22.5	059e0cd8ef78	8 months ago	128MB
k8s.gcr.io/kube-scheduler	v1.22.5	935d3fdcd2d52	8 months ago	52.7MB
k8s.gcr.io/kube-controller-manager	v1.22.5	04185bc88e08	8 months ago	122MB
k8s.gcr.io/kube-proxy	v1.22.5	8f8fdd6672d4	8 months ago	104MB
k8s.gcr.io/nginx-ingress-controller	v1.1.1	c41e9fcadf5a	10 months ago	47.7MB
k8s.gcr.io/etcd	3.5.0-0	004811815584	14 months ago	295MB
nats-streaming	0.22.0	12f2d32e0c9a	14 months ago	19.8MB
k8s.gcr.io/coredns/coredns	v1.8.4	8d147537fb7d	14 months ago	47.6MB
docker/desktop-vpnkit-controller	v2.0	8c2c38aa670e	15 months ago	21MB
docker/desktop-storage-provisioner	v2.0	99f89471f470	15 months ago	41.9MB
k8s.gcr.io/pause	3.5	ed210e3e4a5b	17 months ago	683kB
bolingcavalry/add	latest	5e6c22fa338d	21 months ago	88.4MB
kubernetesui/dashboard	v2.0.0-rc5	fe0df1b24096	2 years ago	126MB
kubernetesui/metrics-scraper	v1.0.3	3327f0dbc64a	2 years ago	40.1MB
prom/alertmanager	v0.18.0	ce3c87f17369	3 years ago	51.9MB

```
PS C:\Windows\system32>
```

### 3.3 启动 k8s

在镜像下载完成后，启动 k8s 前我们还需要修改 `hosts` 文件，增加下面的内容，否则会启动失败。

在路径 `C:\Windows\System32\drivers\etc\hosts` 下用写字本（或其他）方式打开 `hosts` 文件。在最后一行添加下面内容即可：

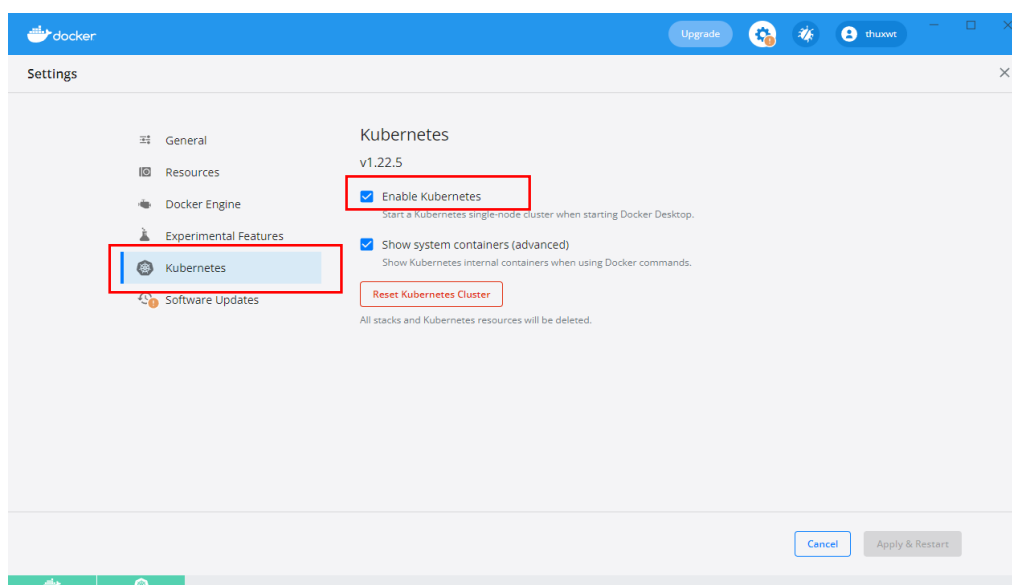
**127.0.0.1   kubernetes.docker.internal**

值得注意的是，通常 `hosts` 文件不能直接修改，可以复制一份在桌面修改完成后替换掉原来的 `hosts` 文件即可。修改后的 `hosts` 文件：

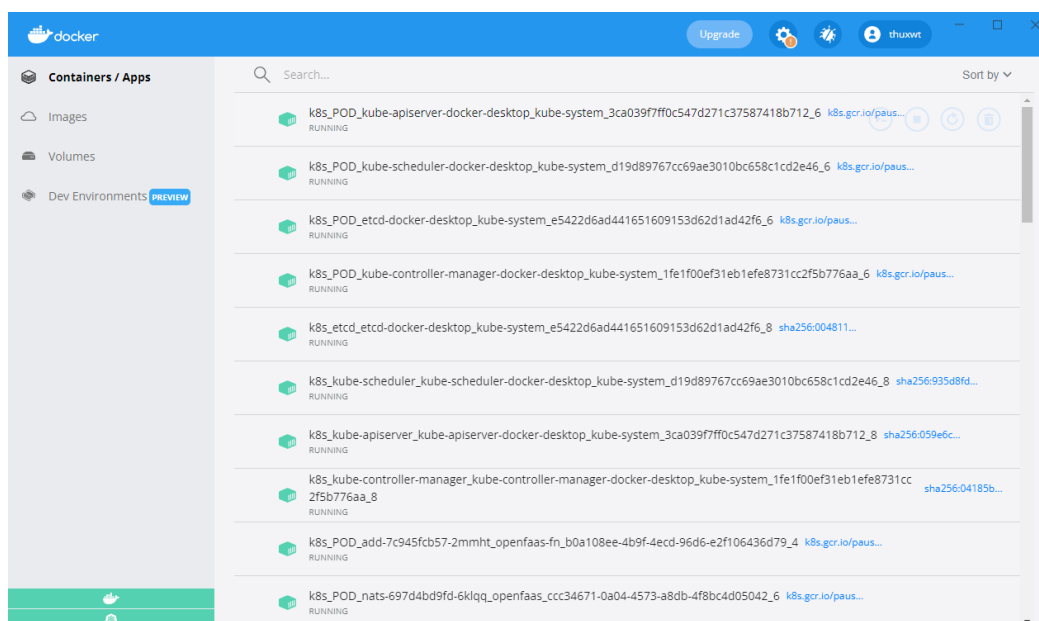
```
hosts - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
# 102.54.94.97   rhino.acme.com   # source server
# 38.25.63.10   x.acme.com       # x client host

# localhost name resolution is handled within DNS itself.
#
# 127.0.0.1     localhost
# ::1           localhost
# Added by Docker Desktop
166.111.73.65 host.docker.internal
166.111.73.65 gateway.docker.internal
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
# End of section
```

此时回到【Dashboard】界面，单击设置，点击左侧【Kubernetes】，选中【Enable Kubernetes】，然后应用。



下面等待 k8s 启动即可，可以通过查看主界面看是否有容器已经启动，如下图所示：



k8s 运行后可以通过下图两个命令查看运行情况：

# 查看集群信息

```
kubectl cluster-info
```

# 获取所有节点信息

```
kubectl get nodes
```



```
PS C:\Windows\system32> 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'.
PS C:\Windows\system32> kubectl get nodes
NAME                STATUS    ROLES    AGE     VERSION
docker-desktop      Ready    control-plane,master   2d21h   v1.22.5
PS C:\Windows\system32>
```

## 4. 配置 Kubernetes 控制台

直接用浏览器打开下面的网址，下载 recommended.yaml 并将其保存在 k8s-for-docker-desktop 文件夹下。

<https://cloud.tsinghua.edu.cn/f/307e1ebf053346769dc8/?dl=1>

之后执行下面的命令：

```
kubectl create -f recommended.yaml
```

会输出如下内容：

```
λ kubectl create -f recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

我们在登陆之前还需要用命令获取到登录所需的 Token：

# 使用下面命令拿到访问令牌

```
kubectl -n kubernetes-dashboard get secret
```

```
PS C:\Windows\system32> kubectl -n kubernetes-dashboard get secret
NAME                                TYPE                                DATA  AGE
default-token-pxtrx                 kubernetes.io/service-account-token 3      2d20h
kubernetes-dashboard-certs          Opaque                               0      2d20h
kubernetes-dashboard-csrf           Opaque                               1      2d20h
kubernetes-dashboard-key-holder     Opaque                               2      2d20h
kubernetes-dashboard-token-hjgrm    kubernetes.io/service-account-token 3      2d20h
```

紧接着我们关注红框中的内容，输入下面的命令获取 Token，注意将命令中的内容和红框中的内容对应



```
kubectl describe secrets -n kubernetes-dashboard kubernetes-  
dashboard-token-hjgrm
```

红框部分就是需要的 **token** 令牌，我们将这个很长的字符串复制，然后打开浏览器输入 <https://localhost:30000/> 访问这个地址，可能会出现以下界面，选择继续前往即可。



集群

Cluster Roles

Namespaces

Nodes

Persistent Volumes

Storage Classes

命名空间

default

概况

工作量

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Namespaces

名字	标签	运行阶段	经过的时间 ↑
<div>✔</div> <div>openfaas-fn</div>	<div>istio-injection: enabled</div> <div>kubernetes.io/metadata.name: openfaas-fn</div> <div>显示所有</div>	Active	an hour
<div>✔</div> <div>openfaas</div>	<div>access: openfaas-system</div> <div>istio-injection: enabled</div> <div>显示所有</div>	Active	an hour
<div>✔</div> <div>kubernetes-dashboard</div>	<div>kubernetes.io/metadata.name: kubernetes-dashboard</div>	Active	an hour
<div>✔</div> <div>default</div>	<div>kubernetes.io/metadata.name: default</div>	Active	an hour
<div>✔</div> <div>kube-node-lease</div>	<div>kubernetes.io/metadata.name: kube-node-lease</div>	Active	an hour
<div>✔</div> <div>kube-public</div>	<div>kubernetes.io/metadata.name: kube-public</div>	Active	an hour
<div>✔</div> <div>kube-system</div>	<div>kubernetes.io/metadata.name: kube-system</div>	Active	an hour

1 - 7 of 7

---

选择左侧的不同的命名空间可以查看不同容器的运行状态,我们后面部署的函数都会被放在名为 `openfaas-fn` 的容器内。

至此, `docker` 和 `k8s` 的安装和配置就已经完成了。

## 5. 安装 OpenFaaS

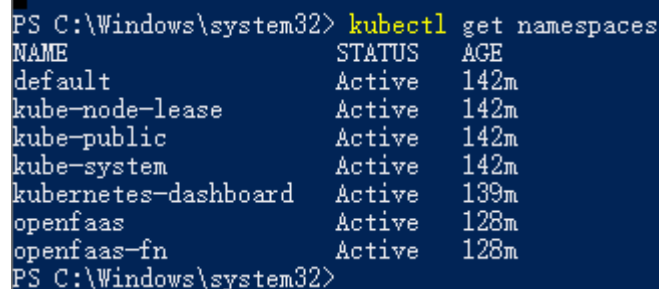
本小节主要介绍 OpenFaaS 的安装和配置。

### 5.1 下载安装 OpenFaaS 并创建命名空间

OpenFaaS 的下载仍然使用 `git` 实现。

```
# 使用下面命令下载 OpenFaaS, 并创建 openfaas 的命名空间
git clone https://github.com/openfaas/faas-netes
cd faas-netes
kubectl apply -f ./namespaces.yml
```

创建完成后, 我们用 `kubectl get namespaces` 可以查看 `k8s` 中的命名空间, 如下图所示, 其中 `openfaas` 为 `openfaas` 组件命名空间, `openfaas-fn` 为部署函数命名空间。



```
PS C:\Windows\system32> kubectl get namespaces
NAME                STATUS    AGE
default             Active   142m
kube-node-lease     Active   142m
kube-public         Active   142m
kube-system         Active   142m
kubernetes-dashboard Active   139m
openfaas            Active   128m
openfaas-fn         Active   128m
PS C:\Windows\system32>
```

### 5.2 创建用户名和密码

为 OpenFaaS 创建用户名和密码是必须要进行的步骤,便于我们对 OpenFaaS 进行管理。命令如下:

```
kubectl -n openfaas create secret generic basic-auth \
--from-literal=basic-auth-user=admin \
--from-literal=basic-auth-password=admin
```

注意, 上面三行代码为一条命令 (“\” 表示连接, 不属于命令)。通过上述命

令，用户名和密码都将被设置为 admin

## 5.3 安装 OpenFaaS 其余组件

这一步直接使用命令行即可完成。具体命令如下：

```
# 进入 faas-netes 文件夹
cd faas-netes
kubectl apply -f ./yaml/
```

安装完成后，我们在命令行输入 `kubectl get pod -n openfaas` 可以验证 OpenFaaS 是否安装完成。最终结果如下则表明安装完成：

```
PS C:\Windows\system32> kubectl get pod -n openfaas
NAME                                READY   STATUS    RESTARTS   AGE
alertmanager-5c757c4488-94r9f      1/1     Running   1 (130m ago)  133m
basic-auth-plugin-85cd598db8-qnkww  1/1     Running   1 (130m ago)  133m
gateway-5b64fb67cb-g9z8r           2/2     Running   4 (129m ago)  133m
nats-697d4bd9fd-nzwhs               1/1     Running   1 (130m ago)  133m
prometheus-bdf79bf9f-nplx9          1/1     Running   1 (130m ago)  133m
queue-worker-8474f87f9d-bprp9       1/1     Running   2 (130m ago)  133m
PS C:\Windows\system32>
```

注：安装 OpenFaaS 后可能需要等待一段时间以上组件的状态才会全部进入 Ready。

## 5.4 安装 faas-cli

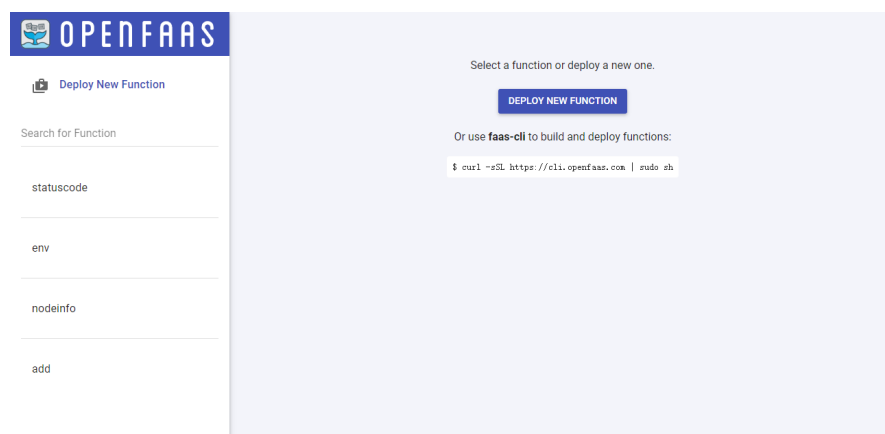
faas-cli 可以帮助我们将编写好的函数打包为 docker 镜像，进而部署到 OpenFaaS 上。Windows 环境下，我们需要下载 faas-cli 的安装包，下载地址：

<https://github.com/openfaas/faas-cli/releases> （下载红框内执行文件即可）

 <a href="#">faas-cli</a>	7.55 MB	10 days ago
 <a href="#">faas-cli-arm64</a>	7.44 MB	10 days ago
 <a href="#">faas-cli-arm64.sha256</a>	81 Bytes	10 days ago
 <a href="#">faas-cli-armhf</a>	6.75 MB	10 days ago
 <a href="#">faas-cli-armhf.sha256</a>	81 Bytes	10 days ago
 <a href="#">faas-cli-darwin</a>	8.1 MB	10 days ago
 <a href="#">faas-cli-darwin.sha256</a>	82 Bytes	10 days ago
 <a href="#">faas-cli.exe</a>	8.07 MB	10 days ago
 <a href="#">faas-cli.exe.sha256</a>	79 Bytes	10 days ago
 <a href="#">faas-cli.sha256</a>	75 Bytes	10 days ago
 <a href="#">Source code (zip)</a>		10 days ago
 <a href="#">Source code (tar.gz)</a>		10 days ago

下载完成后，为了便于操作和管理，建议将 `faas-cli.exe` 执行文件放在 `C:/Windows/System32/faas-cli` 文件夹下。

至此，我们已经成功安装好了 OpenFaaS 和其组件，我们可以通过访问：<https://http://localhost:31112/ui/>访问我们的 OpenFaaS 平台，其中用户名和密码均为上面设置的 `admin`。访问界面如下所示：（右侧为我自行添加的四个函数，可以忽略）



## 6. 注册 Docker 账户，创建自己的 Repository

本节主要介绍在 [hub.docker.com](https://hub.docker.com) 上创建自己的账户，便于我们后续将打包好的镜像上传到 [hub.docker.com](https://hub.docker.com) 上，并在 OpenFaaS 中进行部署。如果已经创建了账户则这一步可以忽略。

创建用户也非常简单，打开 <https://hub.docker.com/>使用邮箱创建一个自己的账户即可，这里不在赘述。需要记住自己的账户名和密码，后续打包、上传镜像需要用到。

## 7. 使用 faas-cli 编写并打包上传自己的函数

本节用一个小小的示例展示如何使用 OpenFaaS 中的 `faas-cli` 工具编写、打包并上传自己的函数。示例所使用的的底层代码为 Python。

### 7.1 下载模板

OpenFaaS 官方提供了编程语言模板，执行命令 `faas-cli template pull`，可以将最新模板下载到本地。

注：需要首先使用命令行进入到 C:/Windows/System32/faas-cli 文件夹下，在该文件夹下需要执行命令 `.\faas-cli template pull`。另外，这一步骤可能会因为网络问题无法连接到 <https://github.com/openfaas/templates>，可以考虑梯子等工具帮助。也可以在云盘上 <https://cloud.tsinghua.edu.cn/d/17a12f0cc36d4b01b578/>打包下载并放到 C:/Windows/System32/faas-cli 文件夹下。

下载完成后，执行命令`\faas-cli new --list`，得到模板如下，可见 OpenFaaS 支持的语言类型还是很丰富的。

```
PS C:\Windows\system32\faas-cli> .\faas-cli new --list
Languages available as templates:
- csharp
- dockerfile
- go
- javall
- javall-vert-x
- node
- node12
- node12-debian
- node14
- node16
- node17
- php7
- php8
- python
- python3
- python3-debian
- ruby
```

## 7.2 创建函数并编写函数

执行以下命令即可创建函数，`add` 是函数名，`python` 是语言类型，`thuxwt` 是在 `hub.docker.com` 上注册的用户名（这里可以换成你自己注册的用户名）

```
.\faas-cli new add -lang python -p thuxwt
```

此时控制台提示如下：

```
Folder: add created.
```

```
/ _ \   _ _      |   |   |   / _ \|   |   |   /  
| | | | '_ \|_ \|'_ \| | | / _ \| / _ \|   |   |\n| | | |_) | |_/ | | | |_) |_) |_) |_) |_) |\n\_\_\_| .__/\_\_\_| | | | |_\_\_, |_\_\_, |_\_\_  
    |_|
```

```
Function created in folder: add  
Stack file written: add.yml
```

在当前目录下产生名为 `add` 的文件夹，以及名为 `add.yml` 的文件。`add.yml` 是函数的描述文件。

进入 `add` 文件夹，看到 `faas-cli` 帮我们生成的源码文件 `handler.py`，将默认代码更改如下：

```
def handle(req):
    array = req.replace('\n', '').split(',')
    rlt = 0
    for i in array:
        rlt += int(i)
    return rlt
```

该函数的作用就是将以逗号隔开的数字字符串的值相加，返回和的值。

下一步回到 `add.yml` 所在目录，执行以下命令开始构建函数：

```
.\faas-cli build -f ./add.yml
```

此后会在本地构建 `docker` 镜像，正常情况下可以构建成功。输入命令 `docker images` 就可以查看本地镜像，其中就有刚创建的 `thuxwt/add`。

```
PS C:\Windows\system32\faas-cli> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
thuxwt/statuscode    latest              d0a0908e16ba       2 days ago         94.4MB
thuxwt/add            latest              9b0cfaf180e8b       2 days ago         87.3MB
```

## 7.3 部署函数

部署函数有两种方式，第一种是自动使用 `faas-cli` 部署，还有一种是利用镜像名称手动部署。下面分别进行介绍

### a) `faas-cli` 自动部署

使用 `faas-cli` 首先需要登录，命令如下：

```
.\faas-cli login -u admin -p admin -g http://localhost:31112/
```

登录成功后显示如下：

```
PS C:\Windows\system32\faas-cli> .\faas-cli login -u admin -p admin -g http://localhost:31112/
WARNING! Using --password is insecure, consider using: cat /faas_pass.txt | faas-cli login -u user --password-stdin
Calling the OpenFaaS server to validate the credentials...
credentials saved for admin http://localhost:31112
```

登录后便可以使用 `faas-cli` 的 `deploy` 部署至 `OpenFaaS`，代码如下：

```
.\faas-cli deploy -f add.yml -g http://localhost:31112/
```

部署成功后显示如下：

```
credentials saved for admin http://localhost:31112
PS C:\Windows\system32\faas-cli> .\faas-cli deploy -f add.yml -g http://localhost:31112
Deploying: add.

Deployed. 202 Accepted.
URL: http://localhost:31112/function/add
```

#### b) 上传镜像手动部署

另外一种方式可以通过 docker 镜像部署。首先使用命令行登录 docker，命令如下：

```
docker login -u username -p password
```

这里的用户名和密码为步骤 6.中注册的用户名和密码。显示 Login Succeeded 则登录成功。接着，我们将镜像打包上传，命令如下：

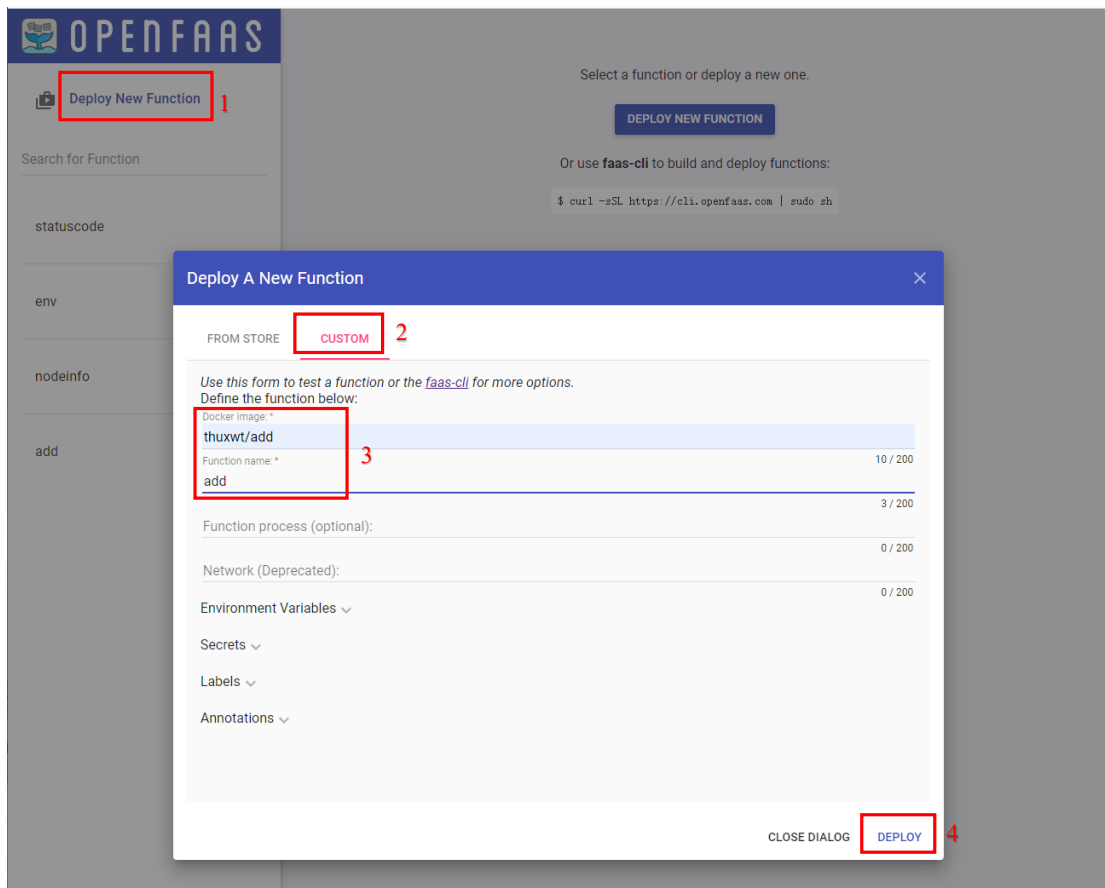
```
docker push thuxwt/add
```

成功上传后，显示如下：

```
promy@ali:~/manager$ docker push thuxwt/add
PS C:\Windows\system32\faas-cli> docker push thuxwt/add
Using default tag: latest
The push refers to repository [docker.io/thuxwt/add]
10b19b15da6f: Layer already exists
73d4f4d85f54: Layer already exists
5f70bf18a086: Layer already exists
4b5b2bb56fbf: Layer already exists
49b68436a526: Layer already exists
982109b25e37: Layer already exists
895b049681bf: Layer already exists
e7545e396b4e: Layer already exists
10b6da389445: Layer already exists
a94165b6a436: Layer already exists
38cdfefb9574b: Layer already exists
40b10d629bfe: Layer already exists
d144d4c5a5eb: Layer already exists
57097237b31b: Layer already exists
532cf93a0d4b: Layer already exists
879c0d8666e3: Layer already exists
20a7b70bdf2f: Layer already exists
3fc750b41be7: Layer already exists
beee9f30bc1f: Layer already exists
latest: digest: sha256:a863168be2a57170363890fdc5ecf81f4aa42a06e9ccbl23c53aace8396073f size: 4693
```

接着，我们访问 <http://localhost:31112>，输入账号和密码（都是 admin）后，进入 OpenFaaS 管理平台。点击左侧的【Deploy New Function】，然后点击上方【CUSTOM】，在 Docker image 中输入镜像名称 thuxwt/add，函数名称可以自己取（这里设置为 add），最后单击【DEPLOY】完成部署。

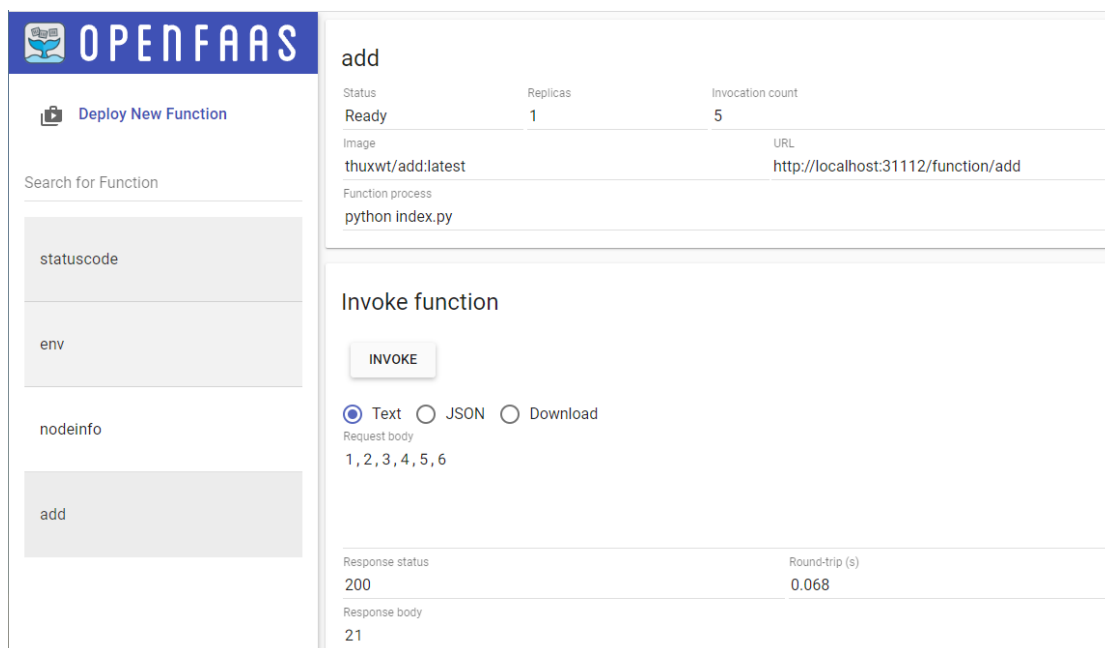




## 7.4 函数测试

部署完成后，我们可以进行简单测试。

- a) 单击右侧 **add**，进入函数。在 **Text** 中输入 ‘1,2,3,4,5,6’ 的字符串，点击 **INVOKE** 可以发现 **Response** 中返回了值 21。



---

b) 使用 Pycharm 或其他 python 编译器，运行以下代码：

```
import requests
url = 'http://localhost:31112/function/add'
response = requests.get(url=url, data='1,2,3,4,55,6,7')
print(response.text)
```

成功输出值 78。

至此，OpenFaaS 的部署结束。