Kustomize使用教程

1. Kustomize是什么

1.1 Kustomize简介

```
    kustomize是一个通过kustomization文件定制kubernetes对象的工具，它可以通过一些资源生成一些新的资源，也可以定制不同的资源的集合，根据各种资源的Gernerator生成对应的资源的yaml，例如configmap、secrets等等···

    kubernetes在1.14版本之后，其内部已集成了kustomize，不需要额外手动安装。

    kustomize在github上目前有8K+star，超300位贡献者，在gitops领域中常常用到。
    github地址：https://github.com/kubernetes-sigs/kustomize
    相关网站：
        https://kustomize.io/
        https://github.com/kubernetes-sigs/kustomize
        https://kubernetes.io/zh/docs/tasks/manage-kubernetes-objects/kustomization/
```

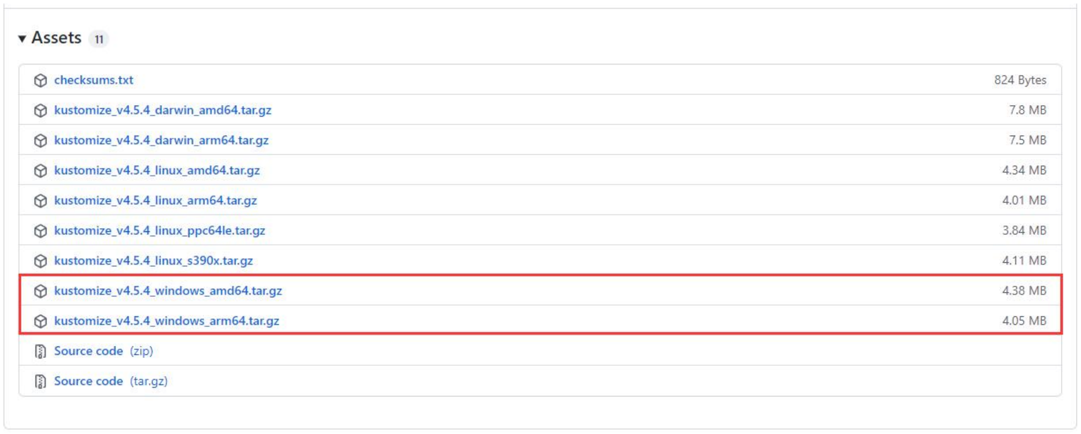
1.2 Kustomize用途

```
    kustomize用途有多种，包含:
    生成资源、全局性字段更改、资源管理提交、资源patch提交等基础使用方法;
    高级的概念和用法有基准（Bases）与覆盖（Overlays）

    实际应用中我们将kustomize分为两类:
    1.基于文件生成新的yaml；2.基于旧yaml进行修改

    实际windows应用中，我们经常都只使用生成新yaml这一个用途，这花费一个篇章进行记录；
    其他的如全局性字段更改、资源管理提交、资源patch提交这些修改旧yaml的都只是添头用途，放在一个章节内记录；
    而修改旧yaml中也有一个比较常用的，就是修改yaml的镜像，在gitops中经常会用到，同时其高级理念base/overlays也在gitops上应用广泛，也会在后文补充记录。
```

2. Windows下使用kustomize生成资源

2.1 文件下载

```
    kustomize在windows下也可以使用，在github上下载windows的kustomize二进制文件。
    下载地址：
        https://github.com/kubernetes-sigs/kustomize/releases/tag/kustomize%2Fv4.5.4
    PS：kustomize_v4.5.4_windows_amd64.tar文件已下载，放置\Kustomize&Helm\Kustomize\windows下
```

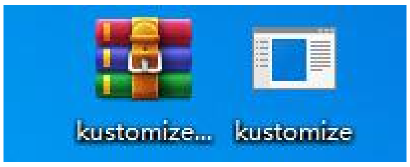


▾ Assets 11

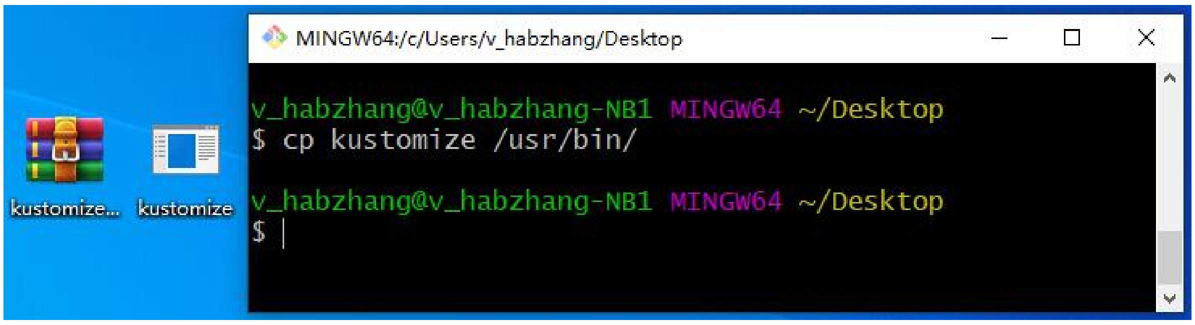
checksums.txt	824 Bytes
kustomize_v4.5.4_darwin_amd64.tar.gz	7.8 MB
kustomize_v4.5.4_darwin_arm64.tar.gz	7.5 MB
kustomize_v4.5.4_linux_amd64.tar.gz	4.34 MB
kustomize_v4.5.4_linux_arm64.tar.gz	4.01 MB
kustomize_v4.5.4_linux_ppc64le.tar.gz	3.84 MB
kustomize_v4.5.4_linux_s390x.tar.gz	4.11 MB
kustomize_v4.5.4_windows_amd64.tar.gz	4.38 MB
kustomize_v4.5.4_windows_arm64.tar.gz	4.05 MB
Source code (zip)	
Source code (tar.gz)	

2.2 文件安装

1）下载的kustomize_v4.5.4_windows_amd64.tar是二进制文件压缩包，解压后为二进制文件；



2）将二进制文件存放在/usr/bin目录下，方便在windows上使用kustomize。



```
v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop
$ cp kustomize /usr/bin/

v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop
$ |
```

2.3 Kustomize使用

windows常用的资源生成器有ConfigMapGenerator、secretGenerator，我们编写kustomization.yaml时指定资源生成器的类型，之后便可引用不同类型的kustomization.yaml生成不同的资源类型。

2.3.1 二进制kustomize相关命令

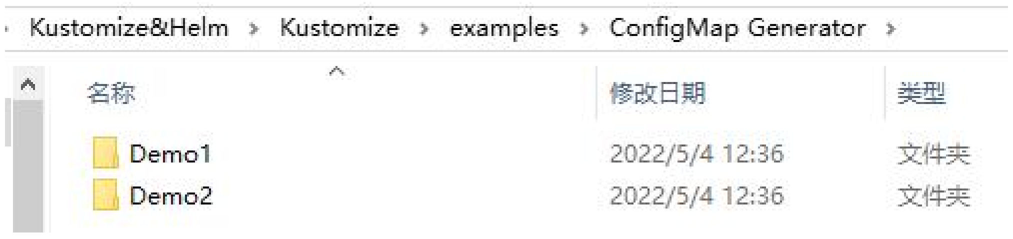
```
$ kustomize.exe --help
Usage:
  kustomize [command]

Available Commands:
  build                      Build a kustomization target from a directory or
URL.
  cfg                        Commands for reading and writing configuration.
  completion                 Generate shell completion script
  create                     Create a new kustomization in the current directory
  edit                       Edits a kustomization file
  fn                         Commands for running functions against
configuration.
  help                       Help about any command
  version                    Prints the kustomize version
```

2.3.2 相关示例文件

```
    PS：示例用到的文件已放置在\Kustomize&Helm\Kustomize\examples文件夹下。
```

2.3.3 ConfigMap Generator

```
    相关文件在\Kustomize&Helm\Kustomize\examples\ConfigMap Generator下
    两个demo：
    demo1是根据已有的一个配置文件生成一份configmap；
    demo2是根据已有的多个配置文件生成一份configmap。
```

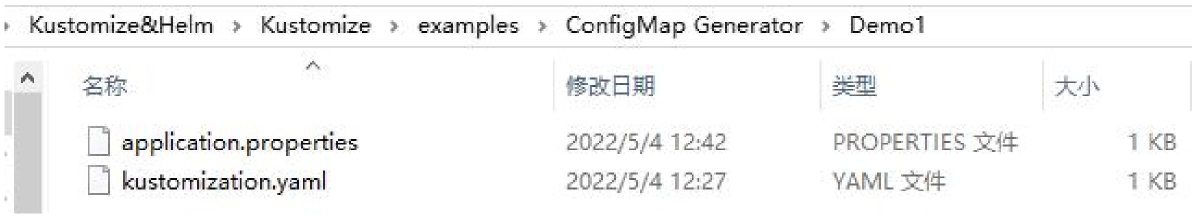


Kustomize&Helm › Kustomize › examples › ConfigMap Generator ›

名称	修改日期	类型
Demo1	2022/5/4 12:36	文件夹
Demo2	2022/5/4 12:36	文件夹

（一）Demo1：单配置文件生成Configmap

```
    一个配置文件：application.properties
    一个kustomization.yaml文件
```



Kustomize&Helm › Kustomize › examples › ConfigMap Generator › Demo1

名称	修改日期	类型	大小
application.properties	2022/5/4 12:42	PROPERTIES 文件	1 KB
kustomization.yaml	2022/5/4 12:27	YAML 文件	1 KB

1）查看已有的单个配置文件

```
$ cat application.properties
FOO=Bar
```

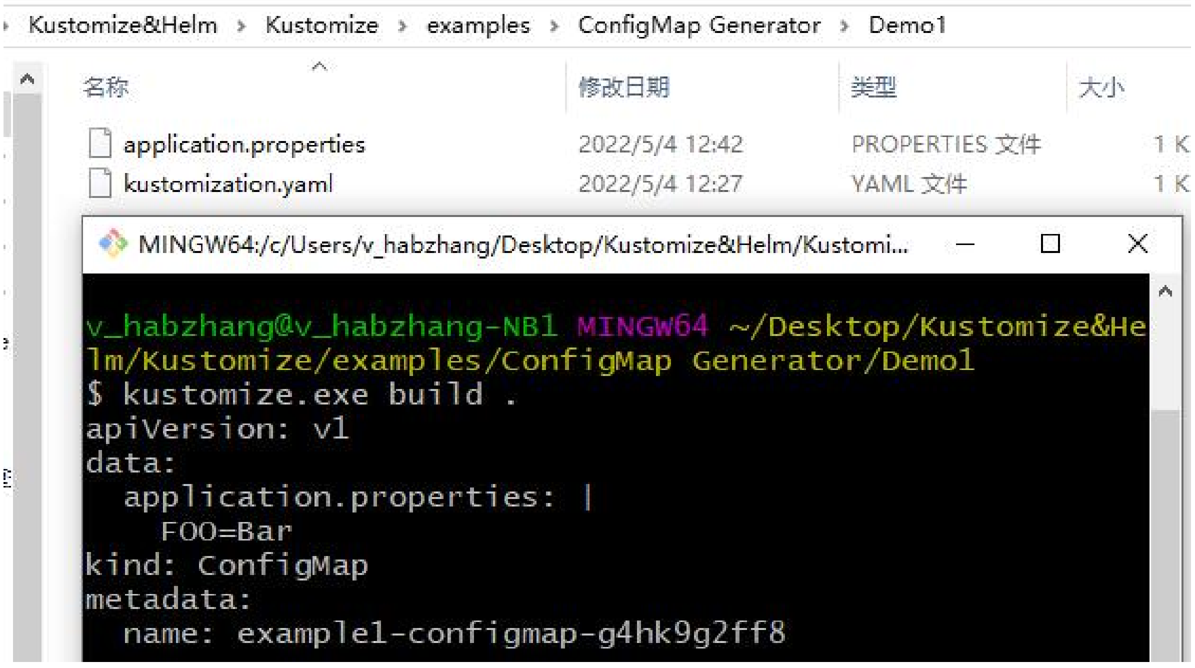
2）查看kustomization.yaml

```
$ cat kustomization.yaml
```

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
- name: example1-configmap
  files:
  - application.properties

PS：kustomization.yaml需要我们填写两个地方：
    1.[configMapGenerator].[name]
    2.[configMapGenerator].[files]

    apiVersion: kustomize.config.k8s.io/v1beta1
    kind: Kustomization
    configMapGenerator:
    - name: 【example1-configmap】#自定义的configmap名字
      files:
      - 【application.properties】#生成configmap时用到的源配置文件名称
```

3）使用kustomize

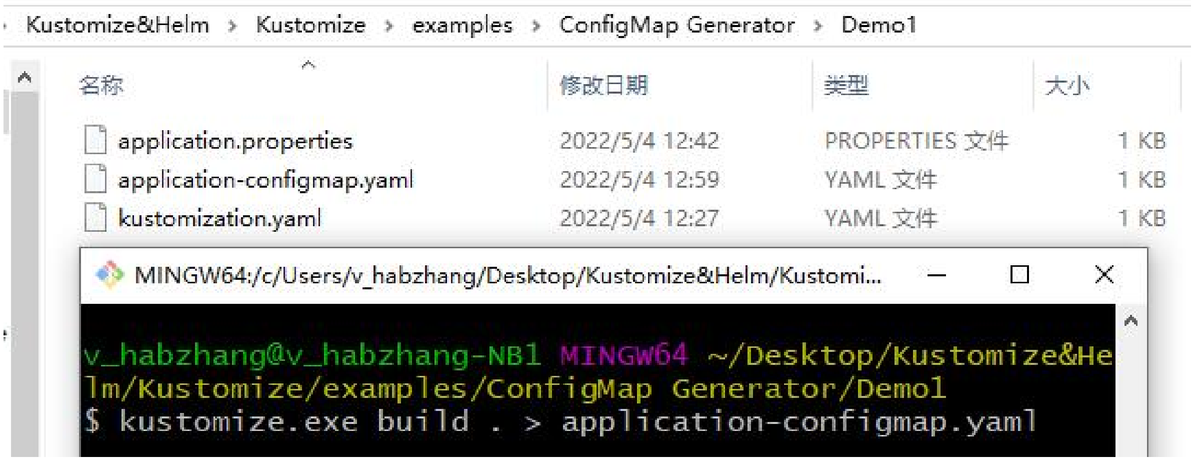
```
同级目录下执行 kustomize build .
看执行完kustomize自动生成configmap的yaml到屏幕

$ kustomize.exe build .
apiVersion: v1
data:
  application.properties: |
    FOO=Bar
kind: ConfigMap
metadata:
  name: example1-configmap-g4hk9g2ff8
```


Kustomize&Helm › Kustomize › examples › ConfigMap Generator › Demo1
名称
修改日期
类型
大小
MINGW64:/c/Users/v_habzhang/Desktop/Kustomize&Helm/Kustomi...

名称	修改日期	类型	大小
application.properties	2022/5/4 12:42	PROPERTIES 文件	1 K
kustomization.yaml	2022/5/4 12:27	YAML 文件	1 K

```
v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop/Kustomize&Helm/Kustomize/examples/ConfigMap Generator/Demo1
$ kustomize.exe build .
apiVersion: v1
data:
  application.properties: |
    FOO=Bar
kind: ConfigMap
metadata:
  name: example1-configmap-g4hk9g2ff8
```

```
也可直接重定向到同级目录下的新文件内
```


Kustomize&Helm › Kustomize › examples › ConfigMap Generator › Demo1
名称
修改日期
类型
大小
MINGW64:/c/Users/v_habzhang/Desktop/Kustomize&Helm/Kustomi...

名称	修改日期	类型	大小
application.properties	2022/5/4 12:42	PROPERTIES 文件	1 KB
application-configmap.yaml	2022/5/4 12:59	YAML 文件	1 KB
kustomization.yaml	2022/5/4 12:27	YAML 文件	1 KB

```
v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop/Kustomize&Helm/Kustomize/examples/ConfigMap Generator/Demo1
$ kustomize.exe build . > application-configmap.yaml
```

4）kustomization.yaml优化

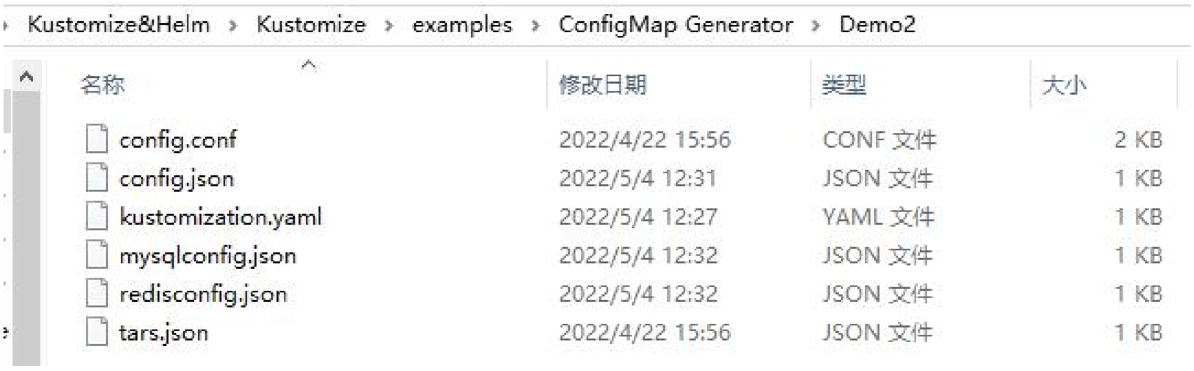
```
    观察kustomize生成的configmap我们发现，该configmap的名字最后边带了一串随机编码："example1-configmap-g4hk9g2ff8"，这是为了区分configmap的版本，所以对名字加了随机编码

$ cat application-configmap.yaml
apiVersion: v1
data:
  application.properties: |
    FOO=Bar
kind: ConfigMap
metadata:
  name: example1-configmap-g4hk9g2ff8


    在实际生产上我们往往不需要名字带随机编码，因此我们可以关闭这一特性，使用我们自定义的名称，在kustomization.yaml添加generatorOptions字段，新的kustomization.yaml如下：

$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
- name: example1-configmap
  files:
  - application.properties
generatorOptions:
  disableNameSuffixHash: true
```

5）使用新kustomization.yaml生成configmap

```
$ kustomize.exe build .
apiVersion: v1
data:
  application.properties: |
    FOO=Bar
kind: ConfigMap
metadata:
  name: example1-configmap

    观察发现该configmap的name不再带随意hash值
```

（二）Demo2：多配置文件生成Configmap

```
    多个配置文件：
    config.conf
    config.json
    kustomization.yaml
    mysqlconfig.json
    redisconfig.json
    tars.json
    一个kustomization.yaml文件
```


Kustomize&Helm › Kustomize › examples › ConfigMap Generator › Demo2

名称	修改日期	类型	大小
config.conf	2022/4/22 15:56	CONF 文件	2 KB
config.json	2022/5/4 12:31	JSON 文件	1 KB
kustomization.yaml	2022/5/4 12:27	YAML 文件	1 KB
mysqlconfig.json	2022/5/4 12:32	JSON 文件	1 KB
redisconfig.json	2022/5/4 12:32	JSON 文件	1 KB
tars.json	2022/4/22 15:56	JSON 文件	1 KB

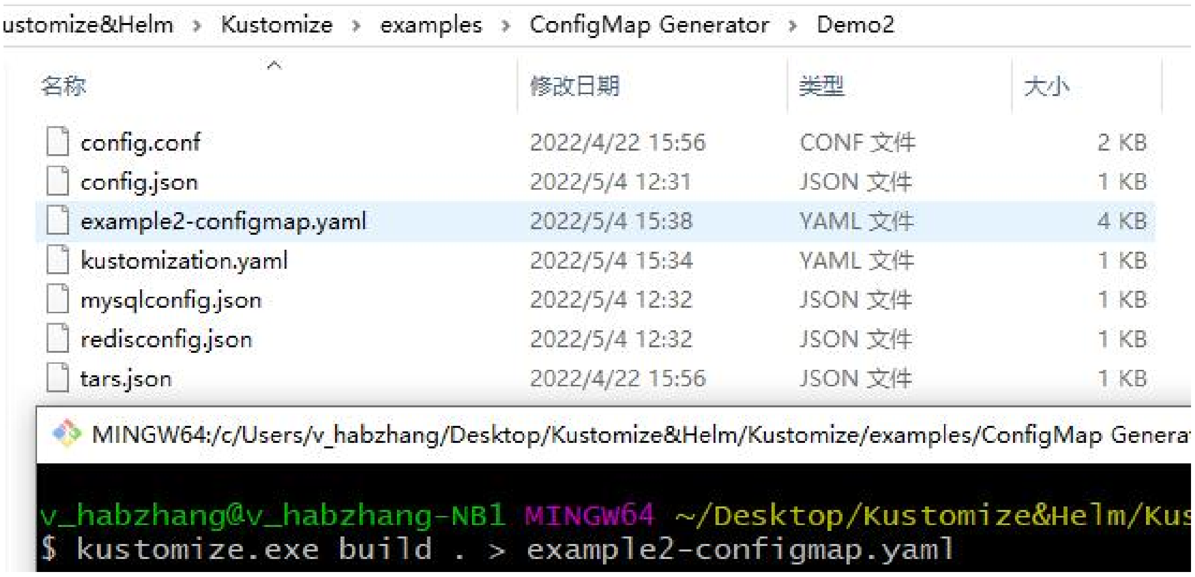
1）查看kustomization.yaml

```
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
- name: example2-configmap
  files:
  - config.conf
  - config.json
  - mysqlconfig.json
  - redisconfig.json
  - tars.json
generatorOptions:
  disableNameSuffixHash: true
```

2）使用kustomize

```
同级目录下执行 kustomize build .
看执行完kustomize自动生成configmap的yaml到屏幕

$ kustomize.exe build .
apiVersion: v1
data:
  config.conf: |
    <tars>
            <application>
                    <server>
                            app=TAdaptor
                            server=AlarmSyncData
                            local=tcp -h 0.0.0.0 -p 10021 -t 30000
                            logpath=/tmp/log
                            <TAdaptor.AlarmSyncData.MainTarsObjAdapter>
                                    allow
                                    endpoint=tcp -h 0.0.0.0 -p 11120 -t 60000

handlegroup=TAdaptor.AlarmSyncData.MainTarsObjAdapter
                                    maxconns=200000
                                    protocol=tars
                                    queuecap=10000
                                    queuetimeout=60000
                                    servant=TAdaptor.AlarmSyncData.MainTarsObj
                                    shmcap=0
                                    shmkey=0
                                    threads=1
                            </TAdaptor.AlarmSyncData.MainTarsObjAdapter>
                            <TAdaptor.AlarmSyncData.CgiObjAdapter>
                                    allow
                                    endpoint=tcp -h 0.0.0.0 -p 8080 -t 60000

handlegroup=TAdaptor.AlarmSyncData.CgiObjAdapter
                                    maxconns=200000
                                    protocol=tars
                                    queuecap=10000
                                    queuetimeout=60000
                                    servant=TAdaptor.AlarmSyncData.CgiObj
                                    shmcap=0
                                    shmkey=0
                                    threads=1
                            </TAdaptor.AlarmSyncData.CgiObjAdapter>
                    </server>
            </application>
    </tars>
  config.json: |-
    {
      "env": "dev",
      "enableZipkin": false,
      "enableModcall": false,
      "deployEnv": "local",
      "gidMappingHost": "TAdaptor.GIDMappingService.GIDMappingObj@tcp -h
tadaptor-gidmapping -p 50092 -t 60000",
      "powerCapacityUrl": "http://10.10.10.10:32515/getRackClm",
      "powerDataHistoryUrl": "http://10.10.10.10/queryHistoryIndicator",
      "disableSyncPowerData": true,
      "disableMdcSyncPowerData": true,
      "mdcSyncOverpowerConfig": {
        "rackConfigCronSpec": "0 */10 * * * *",
        "rackDataCronSpec": "0 */5 * * * *",
        "rackDataMinuteRange": 10,
        "clmConfigCronSpec": "0 */10 * * * *",
        "clmDataCronSpec": "0 */5 * * * *",
        "clmDataMinuteRange": 10
      }
    }
  mysqlconfig.json: |-
    [
      {
        "name": "t_adaptor",
        "host": "10.10.10.10",
        "port": "3306",
        "user": "idc",
        "password": "idc",
        "database": "t_adaptor_326",
        "logLevel": "INFO"
      }
    ]
  redisconfig.json: |-
    [
      {
        "name": "t_adaptor",
        "host": "10.10.10.10",
        "port": "6379",
        "user": "",
        "password": "ceodGpPI8yCbgBtj",
        "database": ""
      },
      {
        "name": "t_adaptor_test",
        "host": "10.10.10.10",
        "port": "6379",
        "user": "root",
        "password": "Aop!@#2014",
        "database": ""
      }
    ]
  tars.json: |-
    {
    }
kind: ConfigMap
metadata:
  name: example2-configmap
```

也可直接重定向到同级目录下的新文件内



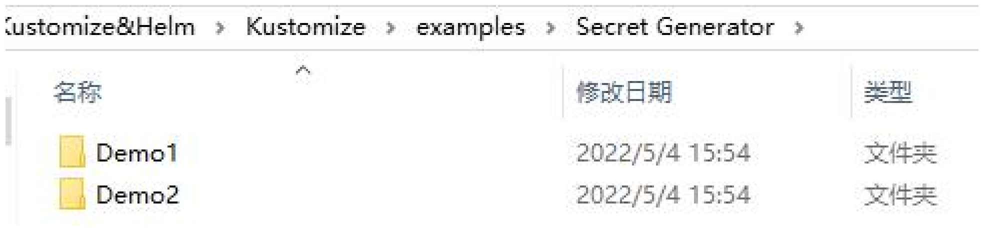
ustomize&Helm › Kustomize › examples › ConfigMap Generator › Demo2

名称	修改日期	类型	大小
config.conf	2022/4/22 15:56	CONF 文件	2 KB
config.json	2022/5/4 12:31	JSON 文件	1 KB
example2-configmap.yaml	2022/5/4 15:38	YAML 文件	4 KB
kustomization.yaml	2022/5/4 15:34	YAML 文件	1 KB
mysqlconfig.json	2022/5/4 12:32	JSON 文件	1 KB
redisconfig.json	2022/5/4 12:32	JSON 文件	1 KB
tars.json	2022/4/22 15:56	JSON 文件	1 KB

```
MINGW64:/c/Users/v_habzhang/Desktop/Kustomize&Helm/Kustomize/examples/ConfigMap Genera

v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop/Kustomize&Helm/Kus
$ kustomize.exe build . > example2-configmap.yaml
```

2.3.4 Secret Generator

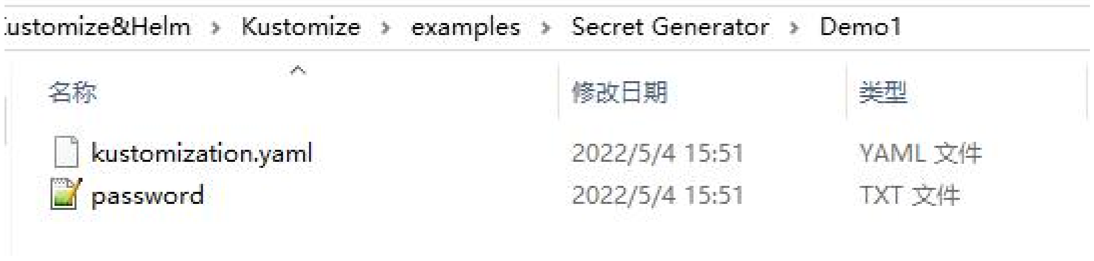
相关文件在\Kustomize&Helm\Kustomize\examples\Secret Generator下
两个demo：
demo1是根据文件生成secret；
demo2是根据键值对生成secret。


Kustomize&Helm › Kustomize › examples › Secret Generator ›

名称	修改日期	类型
Demo1	2022/5/4 15:54	文件夹
Demo2	2022/5/4 15:54	文件夹

（一）Demo1：根据文件生成Secret

```
一个文件：password
一个kustomization.yaml文件
```


Kustomize&Helm › Kustomize › examples › Secret Generator › Demo1

名称	修改日期	类型
kustomization.yaml	2022/5/4 15:51	YAML 文件
password	2022/5/4 15:51	TXT 文件

1）查看password文件

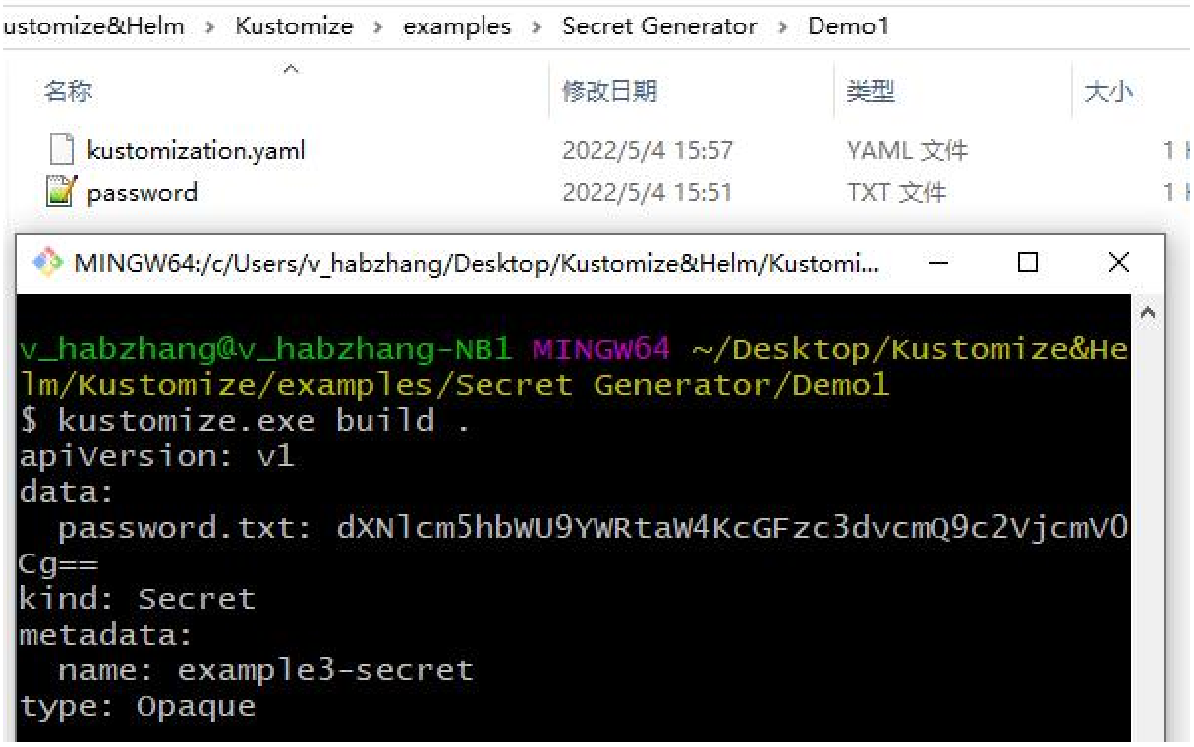
```
$ cat password.txt
username=admin
password=secret
```

2）查看kustomization.yaml

```
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
secretGenerator:
- name: example3-secret
  files:
  - password.txt
generatorOptions:
  disableNameSuffixHash: true

PS：kustomization.yaml同样需要我们填写两个地方：
    1.[configMapGenerator].[name]
    2.[configMapGenerator].[files]

    secretGenerator:
    - name: 【example3-secret】#自定义的secret名字
      files:
      - 【password.txt】#生成secret时用到的源文件名称
```

3）使用kustomize

```
同级目录下执行 kustomize build .
看执行完kustomize自动生成secret的yaml到屏幕

$ kustomize.exe build .
apiVersion: v1
data:
  password.txt: dXNlcm5hbWU9YWRtaW4KcGFzc3dvcmQ9c2VjcmV0Cg==
kind: Secret
metadata:
  name: example3-secret
type: Opaque

PS:Secret资源清单中字段值是Base64编码加密后
的："dXNlcm5hbWU9YWRtaW4KcGFzc3dvcmQ9c2VjcmV0Cg=="，不过，当在Pod中使用Secret时，
kubelet为Pod及其中的容器提供的是解码后的数据
```


ustomize&Helm › Kustomize › examples › Secret Generator › Demo1

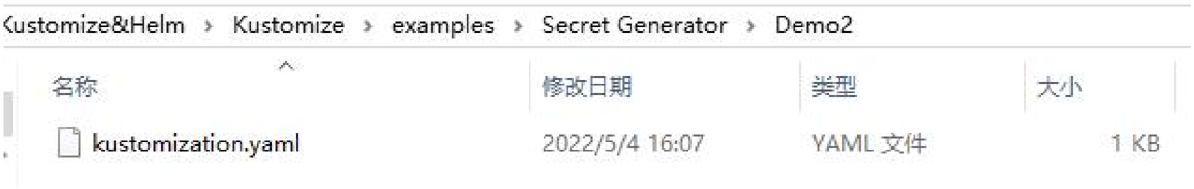
名称	修改日期	类型	大小
kustomization.yaml	2022/5/4 15:57	YAML 文件	1
password	2022/5/4 15:51	TXT 文件	1

MINGW64:/c/Users/v_habzhang/Desktop/Kustomize&Helm/Kustomi...

```
v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop/Kustomize&Helm/Kustomize/examples/Secret Generator/Demo1
$ kustomize.exe build .
apiVersion: v1
data:
  password.txt: dXNlcm5hbWU9YWRtaW4KcGFzc3dvcmQ9c2VjcmV0Cg==
kind: Secret
metadata:
  name: example3-secret
type: Opaque
```

（二）Demo2：根据键值对生成Secret

```
    一个kustomization.yaml文件
```


Kustomize&Helm › Kustomize › examples › Secret Generator › Demo2

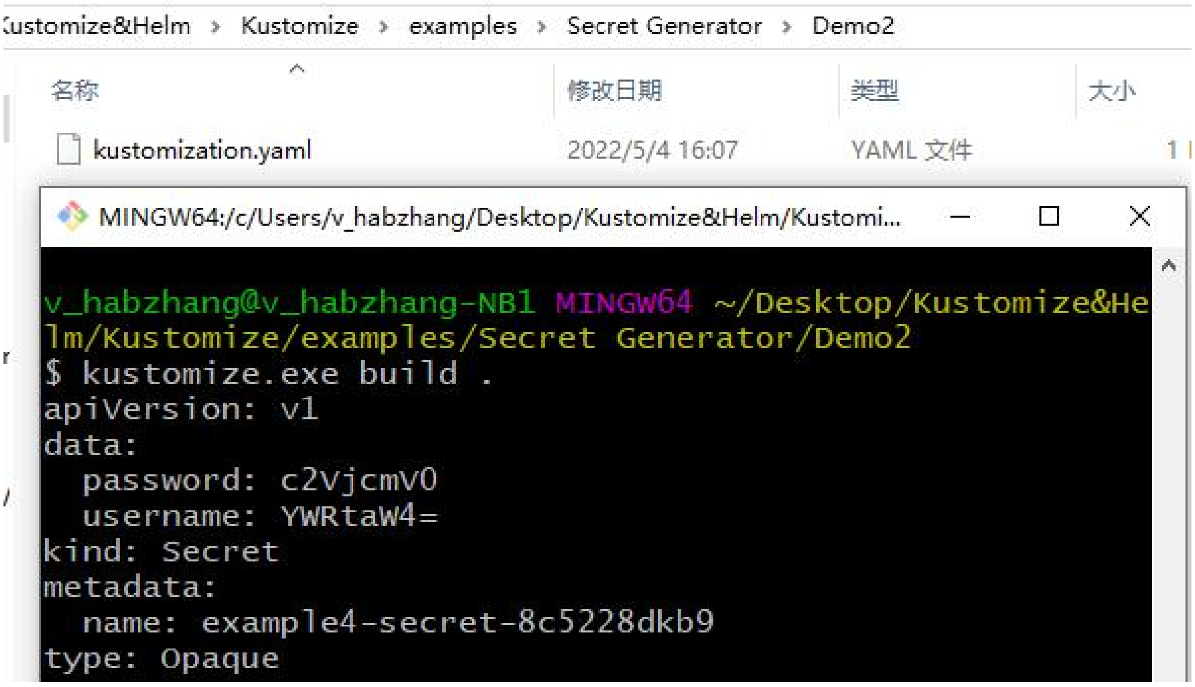
名称	修改日期	类型	大小
kustomization.yaml	2022/5/4 16:07	YAML 文件	1 KB

1）查看kustomization.yaml

```
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
secretGenerator:
- name: example4-secret
  literals:
  - username=admin
  - password=secret
```

2）使用kustomize

```
同级目录下执行 kustomize build .
看执行完kustomize自动生成secret的yaml到屏幕

$ kustomize.exe build .
apiVersion: v1
data:
  password: c2VjcmV0
  username: YWRtaW4=
kind: Secret
metadata:
  name: example4-secret-8c5228dkb9
type: Opaque
```



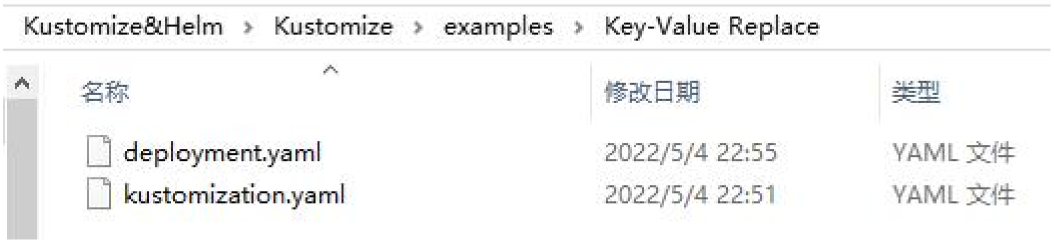
ustomize&Helm › Kustomize › examples › Secret Generator › Demo2
名称
修改日期
类型
大小
kustomization.yaml
2022/5/4 16:07
YAML 文件
MINGW64:/c/Users/v_habzhang/Desktop/Kustomize&Helm/Kustomi...

```
v_habzhang@v_habzhang-NB1 MINGW64 ~/Desktop/Kustomize&Helm/Kustomize/examples/Secret Generator/Demo2
$ kustomize.exe build .
apiVersion: v1
data:
  password: c2VjcmV0
  username: YWRtaW4=
kind: Secret
metadata:
  name: example4-secret-8c5228dkb9
type: Opaque
```

3. Kustomize基于旧yaml进行修改

3.1 全局性字段更改

```
可以实现的功能：
1）替换命名空间
2）为所有对象添加相同的前缀或后缀
3）为对象添加相同的标签集合
4）为对象添加相同的注解集合

相关文件在\Kustomize&Helm\Kustomize\examples\Key-Value Replace下
一个源deployment.yaml
一个kustomization.yaml（定制化修改deployment.yaml的资源清单）
```


Kustomize&Helm › Kustomize › examples › Key-Value Replace

名称	修改日期	类型
deployment.yaml	2022/5/4 22:55	YAML 文件
kustomization.yaml	2022/5/4 22:51	YAML 文件

1）查看已有的deployment资源清单文件

```
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

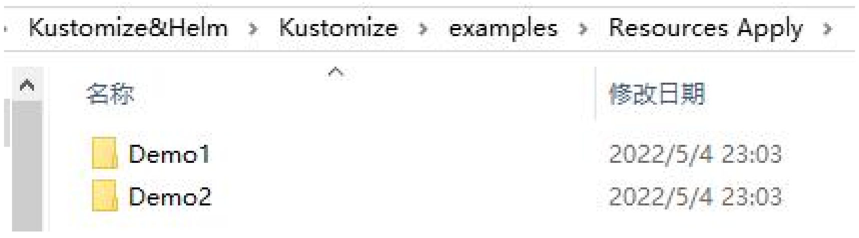
2）使用kustomize进行全局性字段更改

```
1.查看kustomization.yaml清单
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: nginx                    #修改命名空间
namePrefix: ops-                    #namePrefix：为名字添加前缀
nameSuffix: "-001"                  #nameSuffix：为名字添加后缀
commonLabels:                       #添加标签：[key:value]
  organization: littleboy
commonAnnotations:                  #添加注解:[key:value]
  organization-tel: 888-888-8888
resources:                          #源文件
- deployment.yaml
```

```
2.构建新的yaml
$ kustomize.exe build .
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    organization-tel: 888-888-8888
  labels:
    app: nginx
    organization: littleboy
  name: ops-nginx-deployment-001
  namespace: nginx
spec:
  selector:
    matchLabels:
      app: nginx
      organization: littleboy
  template:
    metadata:
      annotations:
        organization-tel: 888-888-8888
      labels:
        app: nginx
        organization: littleboy
    spec:
      containers:
      - image: nginx
        name: nginx

PS：观察发现：
    1.命名空间由default修订为nginx
    2.全局新增注解及标签
    3.deployment名字添加前后缀
```

3.2 资源整合提交&资源patch提交

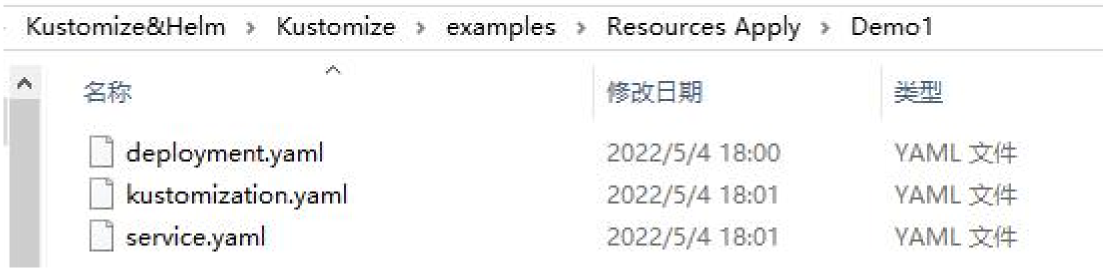
```
    资源整合提交的实质：
    将多个源yaml文件整合到一个yaml中，方便之后一并提交，一次创建所有资源

    资源patch提交的实质：
    对deployment添加某些非注解标签性的字段，例如副本数、资源限制，这些小的改动成为patch（补
丁），再通过这些补丁对yaml进行修改

    相关文件在\Kustomize&Helm\Kustomize\examples\Key-Value Replace下
    两个Demo：
    Demo1是资源整合提交
    Demo2是资源patch提交
```


Kustomize&Helm › Kustomize › examples › Resources Apply ›

名称	修改日期
Demo1	2022/5/4 23:03
Demo2	2022/5/4 23:03

3.2.1 资源整合提交

```
    Demo1文件夹下有三个文件:
    1.service.yaml(nginx的svc清单)
    2.deployment.yaml(nginx的负载清单)
    3.kustomization.yaml
```



Kustomize&Helm › Kustomize › examples › Resources Apply › Demo1

名称	修改日期	类型
deployment.yaml	2022/5/4 18:00	YAML 文件
kustomization.yaml	2022/5/4 18:01	YAML 文件
service.yaml	2022/5/4 18:01	YAML 文件

1）查看service.yaml

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
```

2）查看deployment.yaml

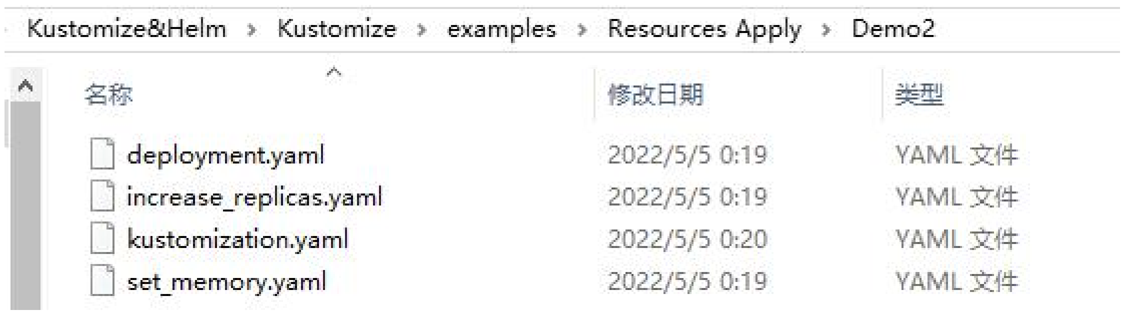
```
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

3）使用kustomize进行全局性字段更改

```
1.查看kustomization.yaml清单
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- service.yaml

2.构建新的yaml
$ kustomize.exe build .
apiVersion: v1
kind: Service
metadata:
  labels:
    run: my-nginx
  name: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - image: nginx
        name: my-nginx
        ports:
        - containerPort: 80
```

3.2.2 资源patch提交

```
    Demo2文件夹下有四个文件：
    1.deployment.yaml(nginx的负载清单)
    2.两个patch文件：increase_replicas.yaml/set_memory.yaml（补丁文件用于定义修改deployment的项）
    3.kustomization.yaml
```


Kustomize&Helm › Kustomize › examples › Resources Apply › Demo2

名称	修改日期	类型
deployment.yaml	2022/5/5 0:19	YAML 文件
increase_replicas.yaml	2022/5/5 0:19	YAML 文件
kustomization.yaml	2022/5/5 0:20	YAML 文件
set_memory.yaml	2022/5/5 0:19	YAML 文件

1）查看deployment.yaml

```
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

2）查看patch文件

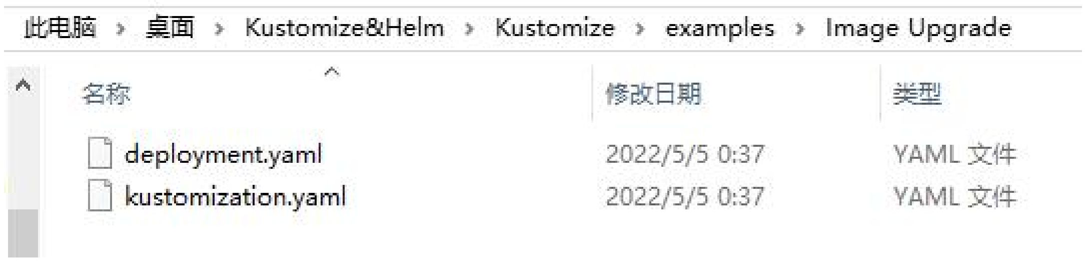
```
1.memory限制的patch清单
$ cat set_memory.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  template:
    spec:
      containers:
      - name: my-nginx
        resources:
          limits:
            memory: 512Mi

2.变更副本数的patch清单
$ cat increase_replicas.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 3
```

3）使用kustomize提交patch构建新yaml

```
1.查看kustomization.yaml
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
patchesStrategicMerge:
- increase_replicas.yaml
- set_memory.yaml

2.构建新yaml
$ kustomize.exe build .
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - image: nginx
        name: my-nginx
        ports:
        - containerPort: 80
        resources:
          limits:
            memory: 512Mi
```

3.3 Deployment镜像修改

```
    kubernetes的cd实质上就是更改服务的镜像版本，因此kustomize的镜像修改在cd过程中会经常用到

    相关文件在\Kustomize&Helm\Kustomize\examples\Image Upgrade下
    一个deployment.yaml
    一个替换镜像的kustomization.yaml
```


此电脑 › 桌面 › Kustomize&Helm › Kustomize › examples › Image Upgrade

名称	修改日期	类型
deployment.yaml	2022/5/5 0:37	YAML 文件
kustomization.yaml	2022/5/5 0:37	YAML 文件

1）查看deployment.yaml

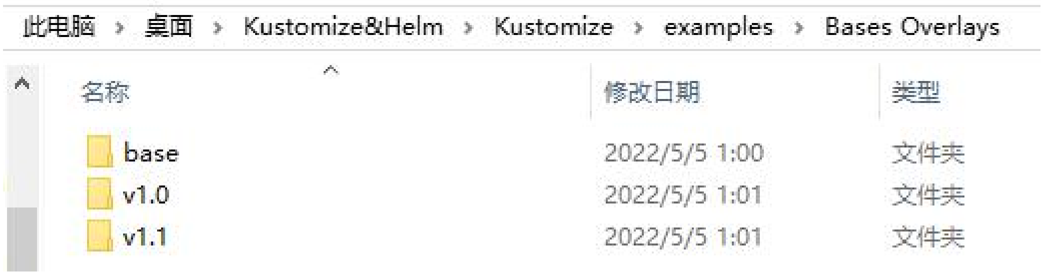
```
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

2）使用kustomize提交patch构建新yaml

```
1.查看kustomization.yaml
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
images:
- name: nginx
  newName: littleboy.registry/nginx
  newTag: 1.4.0

2.构建新yaml
$ kustomize.exe build .
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - image: my.image.registry/nginx:1.4.0
        name: my-nginx
        ports:
        - containerPort: 80
```

4. 基准（Bases）与覆盖（Overlays）

```
	基准Bases是为服务定义一个基础环境（例如初始镜像版本，初始资源限制，服务初始标签等）；
	覆盖Overlays是在基准的基础上执行修改yaml的操作，生成新的环境（例如在初始镜像版本上更改了新的镜像）。
	覆盖Overlays可以有多个不同的，但是每个覆盖Overlays都是基于基准bases进行的。
	
	相关文件在\Kustomize&Helm\Kustomize\examples\Bases Overlays目录下
	其中包含一个基准环境base
	v1.0/v1.1分别是基于基准环境base进行修改的覆盖环境
```

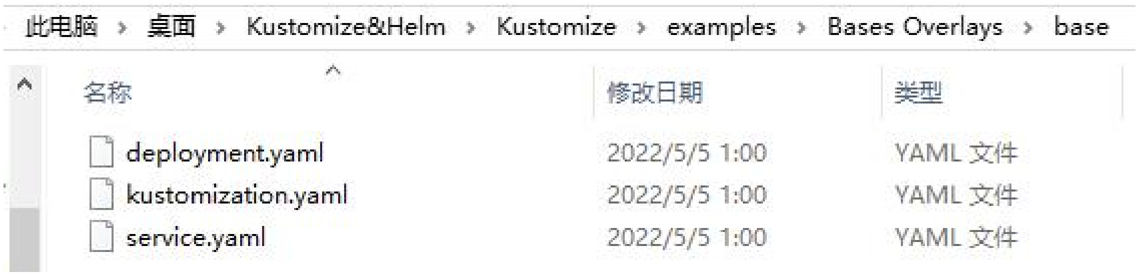


此电脑 › 桌面 › Kustomize&Helm › Kustomize › examples › Bases Overlays

名称	修改日期	类型
base	2022/5/5 1:00	文件夹
v1.0	2022/5/5 1:01	文件夹
v1.1	2022/5/5 1:01	文件夹

(一) 查看基准环境base相关文件

```
	包含服务的service.yaml/deployment.yaml
	以及kustomization.yaml清单
```



此电脑 › 桌面 › Kustomize&Helm › Kustomize › examples › Bases Overlays › base

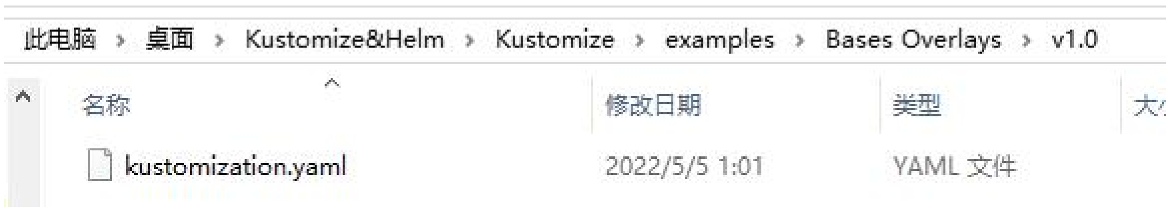
名称	修改日期	类型
deployment.yaml	2022/5/5 1:00	YAML 文件
kustomization.yaml	2022/5/5 1:00	YAML 文件
service.yaml	2022/5/5 1:00	YAML 文件

```
1.查看deployment.yaml
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80

2.查看service.yaml
```

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx

3.查看kustomization.yaml  (资源整合提交)
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- service.yaml
```

(二）查看覆盖overlays环境v1.0/v1.1相关文件

```
    v1.0/v1.1都是基于base进行修改，修改的是deployment的镜像版本，都只有一个
kustomization.yaml
```



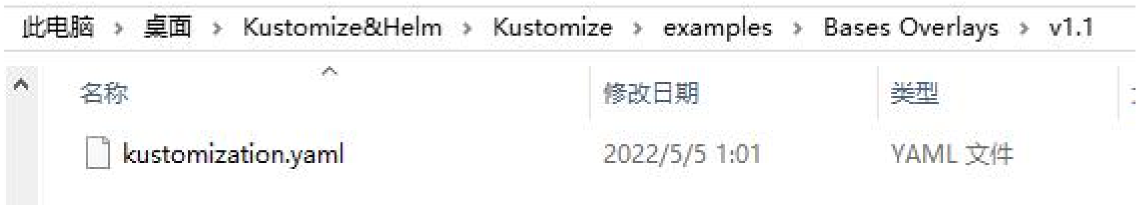
此电脑 › 桌面 › Kustomize&Helm › Kustomize › examples › Bases Overlays › v1.0

名称	修改日期	类型	大
kustomization.yaml	2022/5/5 1:01	YAML 文件	

1）v1.0覆盖环境

```
1.1查看v1.0覆盖环境Overlays的kustomization.yaml
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
bases:
- ../base
images:
- name: nginx
  newName: littleboy.registry/nginx
  newTag: v1.0

1.2.构建新yaml
$ kustomize.exe build .
apiVersion: v1
kind: Service
metadata:
  labels:
    run: my-nginx
  name: my-nginx
spec:
  ports:
```

```
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - image: littleboy.registry/nginx:v1.0
        name: my-nginx
```


此电脑 › 桌面 › Kustomize&Helm › Kustomize › examples › Bases Overlays › v1.1

名称	修改日期	类型
kustomization.yaml	2022/5/5 1:01	YAML 文件

2）v1.1覆盖环境

```
2.1查看v1.1覆盖环境Overlays的kustomization.yaml
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
bases:
- ../base
images:
- name: nginx
  newName: littleboy.registry/nginx
  newTag: v1.1

2.2.构建新yaml
$ kustomize.exe build .
apiVersion: v1
kind: Service
metadata:
  labels:
    run: my-nginx
  name: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
---
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - image: littleboy.registry/nginx:v1.1
        name: my-nginx
```