组件工具调度平台

组件工具打包规范

文档版本：1.2

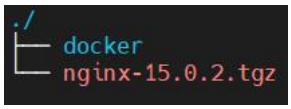
目录

1 容器 Helm 组件工具

1.1 组件工具包格式

Helm 组件工具包中包含 Helm chart 和 container images 两个文件。

- 组件工具包的格式为 tar。
- 使用组件市场提供的离线包构建工具，构建 helm 组件工具离线包。
- 组件工具包结构如下图所示。



上图各个目录文件的说明如下表。

路径及文件	是否必需	说明
docker	是	container images文件夹，此路径下放置容器镜像元数据，通过离线打包工具生成。
nginx-15.0.2.tgz	是	Helm chart包，由离线打包工具生成。此处仅为举例，请以实际为准。

1.2 Helm chart包文件

Helm chart 包中必须包含 Chart.yaml 文件和 values.yaml 两个文件。

1.2.1 Chart.yaml 文件

其中描述了组件工具的基本信息，如果开源组件工具 Chart.yaml 中的信息与组件市场包含信息不一致，请确认至少包含如下组件市场示例内容。

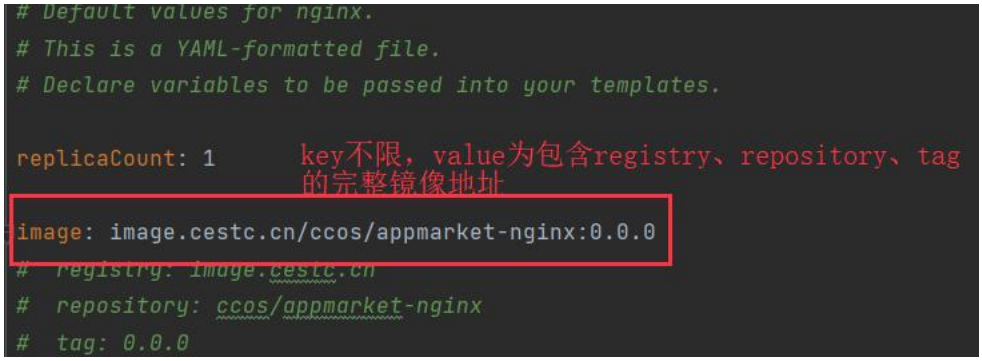
```
annotations:
  arch: x86
  categoryNames: Network
  icon-type: image/png
  vendor: Bitnami
appVersion: 1.16.0
description: A Helm chart for Kubernetes
icon: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
name: sysnginx
version: 0.1.0
```

Chart.yaml 文件参数说明

字段	是否必须	说明
name	是	组件工具名称，长度不超过50个字符。
version	是	组件工具版本，应符合规范，建议使用“1.0.0”“2.0.0”形式命名，不能使用“v0.1非正式版本”或“第1个版本”、“初始版本”这类不规范命名。可使用${VERSION}占位，在使用打包工具构建组件工具包时，将被参数VERSION_TAG动态替换。
arch	是	支持架构，枚举值：x86、X86、arm、ARM、x86/arm。
categoryNames	是	组件工具分类，可填多条，填写分类的英文名称，组件工具分类信息请参见组件工具分类。
vendor	是	组件工具包服务商。
icon	是	组件工具Logo，必填，支持png格式，图片base64编码后填入此字段。
icon-type	是	组件工具Logo格式，必填，目前支持image/png
description	是	组件工具说明，至少包含组件工具详细信息、组件工具的核心功能和服务、各个组件工具规格的功能和服务等。

1.2.2 values.yaml 文件

- values.yaml 文件中需要添加 relatedImages 字段，用于记录 helm chart 会使用到的容器镜像。
- relatedImages 中可添加多个容器镜像信息。
- chart 包中所有使用容器镜像的 manifests 均需要从 values.yaml 中动态获取容器镜像地址。
- values.yaml 中所有对容器镜像的动态配置都由一个包含 registry、repository、tag 三部分的完整镜像地址字段配置。
- values.yaml 文件中所有使用到容器镜像的 registry 配置将被替换为离线镜像仓库地址。
- values.yaml 文件 relatedImages 参数块配置模板如下，请根据表格修改对应参数。



values.yaml 文件 relatedImages 参数块说明

字段	是否必须	说明
relatedImages	是	用于记录helm chart会使用到的容器镜像，可以添加多个容器镜像，每个容器镜像都由一个包含registry、repository、tag三部分的完整镜像地址字段配置。镜像tag可使用${VERSION_TAG}占位，在使用打包工具构建组件工具包时，将被参数VERSION_TAG动态替换。

1.3 打包工具使用说明

打包工具 appmarket-package-helm 是由 golang 实现并编译生成的二进制文件。用于构建符合组件市场规范的 Helm 组件工具包，包含离线镜像元数据及 helm chart 包。运行在 x86 架构的 podman 容器环境中。

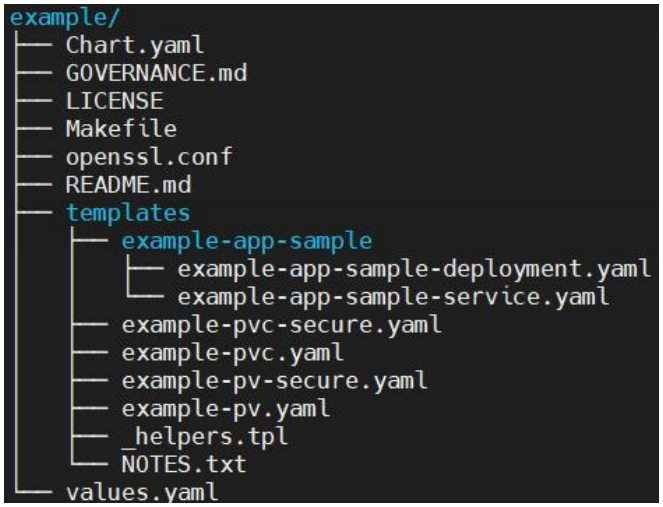
1.3.1 组件工具包构建环境

- 准备组件工具包构建环境，架构 X86，centos 操作系统；
- 组件工具包构建环境有 podman，能够 run 容器镜像；
- 能够访问容器镜像所在镜像仓库，并拉取镜像。

1.3.2 使用方法

1.3.2.1 环境准备

- 将包含 helm chart 包内容的文件目录拷贝至组件工具包构建环境；



- 配置本地 docker-registry 运行环境，执行以下命令

```
mkdir -p /opt/registry/{auth,certs,data}
cp registry.key /opt/registry/certs/registry.key
cp registry.crt /opt/registry/certs/registry.crt
cp   root-ca.crt /opt/registry/certs
cp -f root-ca.crt   /etc/pki/ca-trust/source/anchors/   && update-ca-trust
echo "10.255.89.132 image.ccos.io" >> /etc/hosts （IP 使用组件工具包构建环境本机 IP）
cp htpasswd   /opt/registry/auth
```

- 将文件 podman 拷贝至/usr/bin 目录下，并赋权

```
chmod 755 /usr/bin/podman
```

- 拷贝 docker-registry.tar 至组件工具包构建环境，并执行命令

```
podman load -i docker-registry.tar
```

- 运行本地 docker-registry 镜像仓库，执行如下命令

```
podman run --name mirror-registry -p 443:5000 \
    -v /opt/registry/data:/var/lib/registry:z \
    -v /opt/registry/auth:/auth:z \
    -e "REGISTRY_AUTH=htpasswd" \
    -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
    -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
    -v /opt/registry/certs:/certs:z \
    -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/registry.crt \
    -e REGISTRY_HTTP_TLS_KEY=/certs/registry.key \
    -e REGISTRY_COMPATIBILITY_SCHEMA1_ENABLED=true \
    -d image.cestc.cn/ccos-ceake/registry:1.0
```

- 确认可以访问本地 docker-registry

```
curl -u lyb:123456 -k https://image.ccos.io /v2/_catalog
```

- 修改 all-secret.json，添加镜像源仓库访问认证信息

“image.cestc.cn”为举例，将“image.cestc.cn”修改为镜像源镜像仓库认证信息。

“auth”字段为镜像源镜像仓库用户名:密码(user:password)base64 编码

“image.ccos.io”不要修改，为本地启动 docker-registry 认证信息。

- 将文件 ccos 拷贝至/usr/bin 目录下，并赋权

```
chmod 755 /usr/bin/ccos
```

- 将文件 helm 拷贝至/usr/bin 目录下，并赋权

```
chmod 755 /usr/bin/helm
```

1.3.2.2 构建组件工具包

- 执行如下命令运行打包工具 appmarket-package-helm

```
appmarket-package-helm "${REGISTRY_REPO}" "${VERSION_TAG}" "${ARCH}" "${PACKAGE_FULLNAME}" "${PACKAGE_PATH}" "${REGISTRY_PATH}" "${CODE_PATH}" "${REGISTRY_AUTH_FILE}" "${WORK_TMP_PATH}" "${PREVIOUS_VERSION}" "${PACKAGE_TYPE}"
```

使用时需要指定相应参数，参数说明如下：参数全部必填

序号	参数	说明
1	REGISTRY_REPO	离线镜像repo
2	VERSION_TAG	版本号
3	ARCH	组件工具包架构，即组件工具包中包含容器镜像支持的架构。可传枚举值： amd64、arm64
4	PACKAGE_FULLNAME	组件工具包名
5	PACKAGE_PATH	构建完成后组件工具包存放位置
6	REGISTRY_PATH	docker-registry数据目录，固定传入“/opt/registry/data/”
7	CODE_PATH	Chart包元数据目录

8	SOURCE_REGISTRY_CERT	镜像源仓库的访问认证文件，即all-secret.json
9	WORK_TMP_PATH	临时工作目录，固定传入“”
10	IMAGE_LIST_PATH	镜像清单文件存放位置，固定传入“”
11	PREVIOUS	前序版本号，固定传入“X”
12	PACKAGE_TYPE	组件工具包类型，固定传入“helm”

举行如下，执行：

```
./appmarket-package-helm example test0.1 amd64 test_package_name_0.1 /home/test/package /opt/registry/data/ ./example ./all-secret.json "" "" "X" helm
```

- 执行以上命令后，将会构建一个版本为 test0.1，包名为 test_package_name_0.1_x86_64.tar，适用于 x86 环境的 Helm 组件工具包；若 ARCH 传入 arm64，构建后的包名为 test_package_name_0.1_aarch64.tar
- 构建完成后组件工具包将存放在/home/test/package 目录下；
- Chart 元数据在与打包工具二进制文件 appmarket-package-helm 同路径下的 example 目录中

1.4 文件清单

提供使用打包工具本地构建组件工具包的相关文件 package.zip，请解压后使用。

文件清单如下：

文件名	说明
appmarket-package-helm	打包工具二进制文件，请参照构建组件工具包中的说明使用
podman	cos命令二进制文件，请参照环境准备中的说明使用
ccos	ccos命令二进制文件，请参照环境准备中的说明使用
helm	helm命令二进制文件，请参照环境准备中的说明使用
docker-registry.tar	本地docker-registry容器镜像文件，，请参照环境准备中的说明使用
registry.crt	本地docker-registry证书文件，请参照环境准备中的说明使用
registry.key	本地docker-registry访问私钥文件，请参照环境准备中的说明使用
httppasswd	本地docker-registry访问认证文件，请参照环境准备中的说明使用
all-secret.json	镜像源仓库、本地docker-registry访问认证文件，请参照环境准备、构建组件工具包中的说明使用
root-ca.crt	root-ca证书文件，请参照环境准备中的说明使用

2 容器 Operator 组件工具

2.1 组件工具包格式

Operator 组件工具包，是一个 tar.gz 压缩文件，内部包含 CatalogSource.yaml 和 image.tar 两个文件。而 image.tar 文件内部又包含 index image、bundle image、operator image、operand image 的镜像。

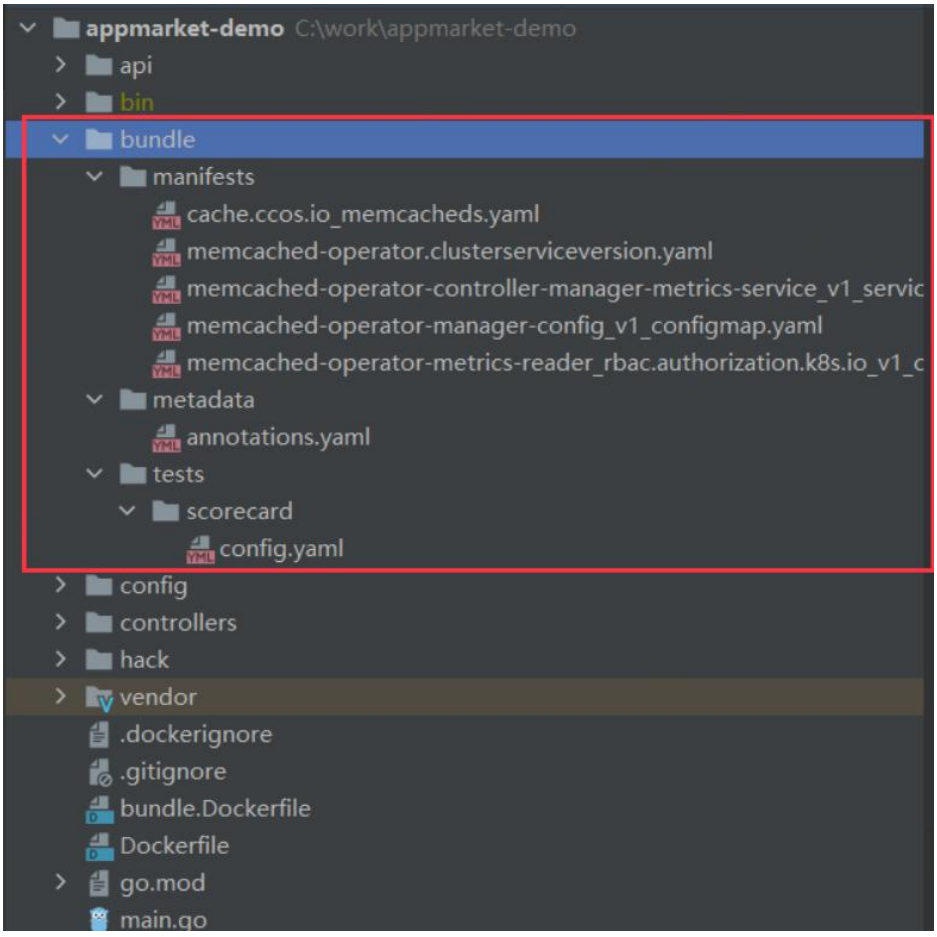
- 使用组件市场提供的离线包构建工具，构建 Operator 组件工具离线包。

2.2 Operator组件应用包文件

2.2.1 bundle image

对 bundle 镜像的打包，有如下规范：

- bundle 目录结构遵循 operator-sdk 创建出来的项目结构，其位于 operator 项目根目录下的 bundle 目录，其下有 manifests、metadata、tests 三个子目录。目录结构如下所示：



- 使用统一的 Dockerfile 完成 bundle 镜像的构建，Dockerfile 如下所示：

```
FROM scratch

# Core bundle labels.
LABEL operators.operatorframework.io.bundle.mediatype.v1=registry+v1
LABEL operators.operatorframework.io.bundle.manifests.v1=manifests/
LABEL operators.operatorframework.io.bundle.metadata.v1=metadata/
LABEL operators.operatorframework.io.bundle.package.v1=memcached-operator
LABEL operators.operatorframework.io.bundle.channels.v1=alpha
LABEL operators.operatorframework.io.metrics.builder=operator-sdk-v1.10.1
LABEL operators.operatorframework.io.metrics.mediatype.v1=metrics+v1
LABEL operators.operatorframework.io.metrics.project_layout=go.kubebuilder.io/v3

# Labels for testing.
LABEL operators.operatorframework.io.test.mediatype.v1=scorecard+v1
LABEL operators.operatorframework.io.test.config.v1=tests/scorecard/

# Copy files to locations specified by labels.
COPY bundle/manifests /manifests/
COPY bundle/metadata /metadata/
COPY bundle/tests/scorecard /tests/scorecard/
```

2.2.2 index image

一个 index image 可以包含一个或者多个 operator 应用。因此，与 bundle image 的打包不同，bundle image 是在 operator 的代码仓库中打包，而 index image 打包是在一个独立的代码仓库中。index image 内部的 operator.yaml 内容如下：

```
Schema: olm.semver
GenerateMajorChannels: true                    # 是否将 major 的版本号作为升级路径名字的一部分
GenerateMinorChannels: false                   # 是否将 minor 的版本号作为升级路径名字的一部分
Stable:                                        # 升级路径的名字
  Bundles:                                     # 升级路径包含的 operator 版本
    - Image: quay.io/foo/olm:testoperator.v0.1.0      # operator 的 bundle image 地址
```

2.2.3 CatalogSource

CatalogSource 相关规范如下：

- 文件名必须为"CatalogSource.yaml"
- 建议升级是包中不要修改部署包中的 CatalogSource.yaml 文件，尤其其名字（metadata.name），否则可能导致无法升级应用
- CatalogSource 中的 metadata.namespace 必须与应用 bundle image 中 clusterserviceversion.yaml 的 metadata.annotations.operatorframework.io/suggested-namespace 保持一直，即与应用推荐命名空间一致
- CatalogSource 中的 spec.sourceType 必须指定为 grpc

CatalogSource.yaml 的示例如下：

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: memcached-operators
  namespace: ccos-memcached
spec:
  displayName: memcached operators
  image: image.cestc.cn/ccos/memcached-operators:v0.1
  publisher: cecloud
  sourceType: grpc
  updateStrategy:
    registryPoll:
      interval: 10m
```

2.3 打包工具使用说明

operator 组件的打包工具和 helm 组件的打包工具是一致的，唯一的区别是调用打包工具时不需要传 PACKAGE_TYPE 参数

3 附录 - 组件工具分类

组件工具分类包括一级分类和二级分类，在压缩组件工具包时，需填写二级分类的英文名称。若未发现有合适的二级分类，可填写一级分类的英文名称，最终会将其分配到一级分类的“其他”子分类中。

组件工具的默认分类信息如下表所示，支持按需添加和更改分类。

一级分类（英文）	二级分类	二级分类英文	Code
计算（Computing）	云服务器	Cloud Server	101
	高性能计算	High Performance Computing	102
	Serverless	Serverless	103
	操作系统	Operating System	104
	边缘计算	Edge Computing	105
容器（Container)	容器服务	Container Service	201
存储（Storage）	基础存储服务	Basic Storage Service	301
	存储数据服务	Storage Data Service	302
	数据迁移与工具	Data Migration&Tools	303
	混合云存储	Hybrid Cloud Storage	304
网络与CDN（Network&CDN）	云上网络	Cloud Based Networking	401
	跨地域网络	Cross Region Networking	402
	混合云网络	Hybrid Cloud Networking	403
	CDN	CDN	404
数据库（Database）	关系型数据库	Relational Database	501
	NoSQL数据库	NoSQL Database	502
	数据仓库	Data Warehouse	503
	数据库管理工具	Database Management Tools	504
大数据（BigData）	数据计算与分析	Data Computing & Analytics	601
	数据湖	Data Lake	602
	数据应用与可视化	Data Applications & Visualization	603
	数据开发与服务	Data Development & Services	604

一级分类（英文）	二级分类	二级分类英文	Code
中间件（Middleware）	微服务工具与平台	Microservices Tools & Platforms	701
	云消息队列	Cloud Message Queue	702
	云原生可观测	Cloud Native Observability	703
	应用集成	Application Integration	704
人工智能（AI）	自然语言处理	Natural Language Processing	801
	文本	Text	802
	图像	Image	803
	计算机视觉	Computer Vision	804
	视频	Video	805
	音频	Audio	806
	生成式AI	Generative AI	807
	数据标注	Data Labeling	808
安全（Security）	主机安全	Machine Security	901
	网络安全	Network Security	902
	数据安全	Data Security	903
	Web应用防火墙	Web Application Firewall	904
	堡垒机	Bastion Host	905
物联网（IoT）	智能硬件	Intelligent Hardware	1001
	模组	Modules	1002
	传感器	Sensor	1003
	集成系统	Integration System	1004
开发和运维（Development&Operation）	监控	Monitoring	1101
	日志	Log	1102
	测试	Testing	1103
	过程管理	Process Management	1104
	源码控制	Source Control	1105
	问题跟踪	Issue Tracking	1106

一级分类（英文）	二级分类	二级分类英文	Code
解决方案（Solution）	内容管理	Content Management	1201
	IT管理	IT Management	1202
	办公协同	Collaborative Working	1203
	人事管理	Human Resources	1204
	财税管理	Financial Services	1205
	身份管理	Identity Management	1206