

27 | 开发互不干扰，如何实现自动多环境管理？

2023-02-08 王伟 来自北京

《云原生架构与GitOps实战》

课程介绍 >



讲述：王伟

时长 09:30 大小 8.68M



你好，我是王伟。

从这节课开始，我们开始学习 GitOps 的多环境管理和安全方面的内容。

聊起多环境，你可能会立即想到下面几个常见的环境：

- 开发环境
- 测试环境
- 预发布环境
- 生产环境

为了让不同职责的人员在不同的环境下独立工作，我们一般会将不同环境隔离。通常，开发环境主要用于开发人员的日常开发，测试环境则是为测试团队而准备的，预发布是正式发布到生

产环境之前的最后一道防线，除了数据以外，应该尽量和生产环境保持一致。

当然，对有些团队来说，他们可能还希望开发人员之间相互隔离，也就是为每一个开发者分配一个独立的开发环境，使他们互不干扰。

在非云原生技术架构体系下，环境一般是由特定的团队人工维护的。所以，要想得到一个新的环境，由于文档和技术方面的原因，过程并不简单。但是，在云原生的业务架构体系下，应用是通过标准的 **Kubernetes** 对象被“定义”出来的。所以，在这种情况下，得到一个新的环境就变得非常容易了。

在之前介绍的 **GitOps** 工作流中，我们都是以部署单个环境为例子的。那么，如果我希望为同一个应用创建新的环境，甚至是为不同的开发者创建隔离的开发环境，怎么做才最合适呢？除了手动创建重复的 **ArgoCD** 应用，还有没有更好的技术方案？

这节课，我们来看看如何使用 **ArgoCD ApplicationSet** 来实现 **GitOps** 自动多环境管理，并通过 **ArgoCD Generator** 来达到“**代码即环境**”的效果。

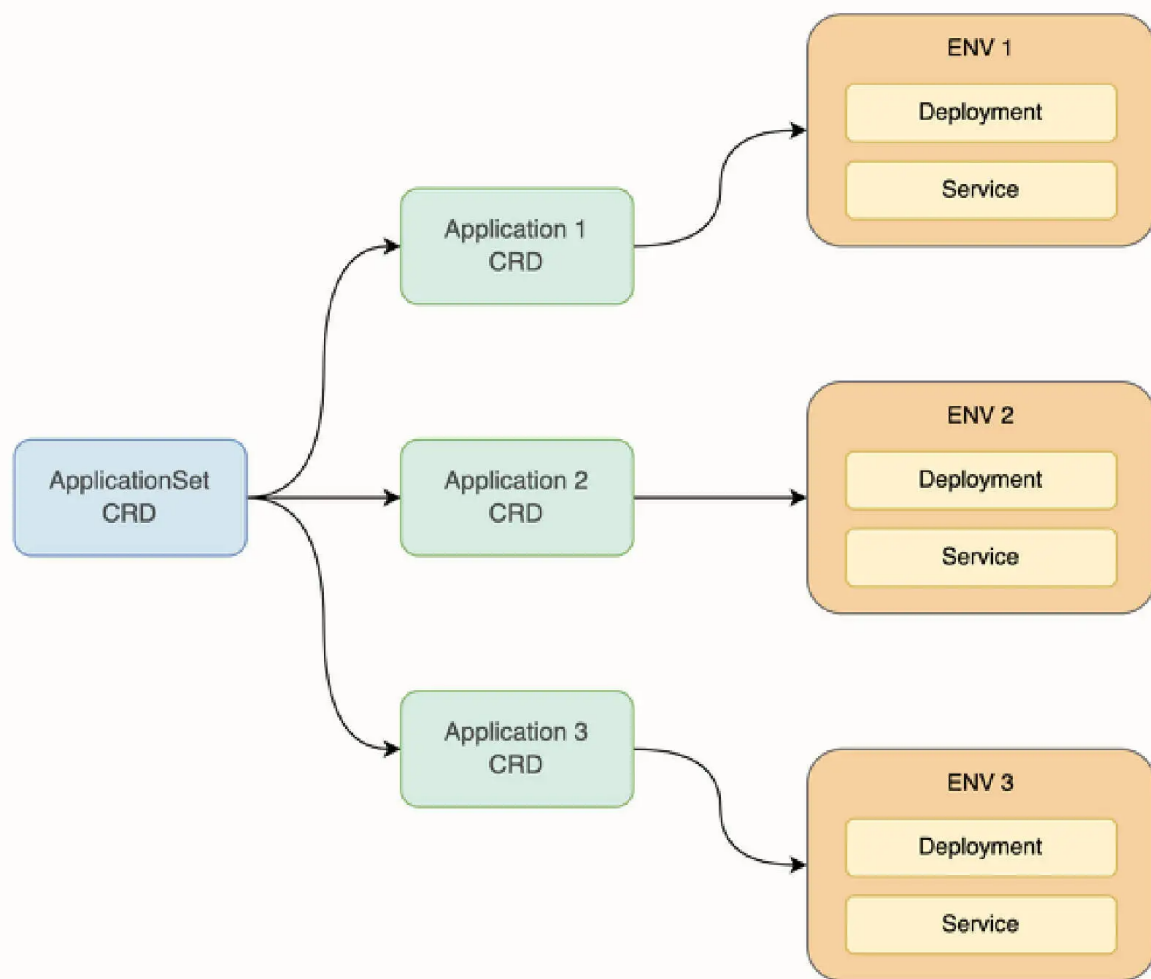
在开始之前，你需要在本地的 **Kind** 集群安装下面两个组件。

- 安装 **ArgoCD**。
- 安装 **Ingress-Nginx**。

此外，你还需要克隆 [🔗 示例仓库](#)，并将它推送到你的 **Git** 仓库中。

自动多环境管理概述

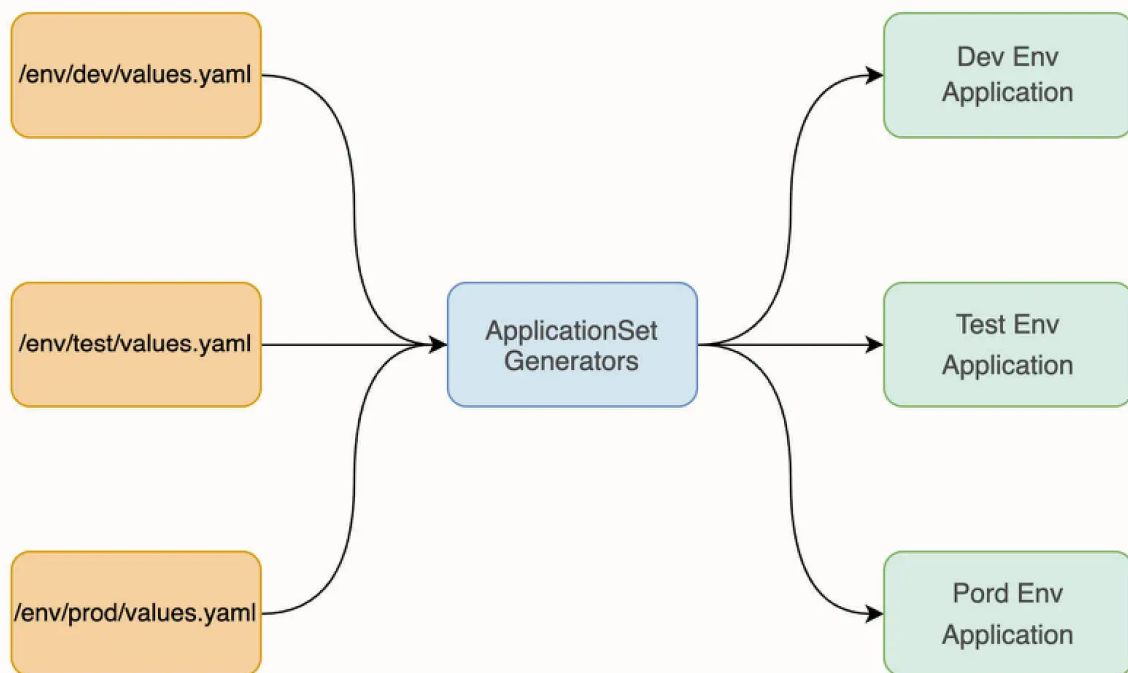
在正式进入实战之前，我们先来了解一下使用 **ArgoCD ApplicationSet** 来实现自动多环境管理的整体架构，如下图所示。



在这张架构图中，我们会创建一个 **ApplicationSet** 对象，它是一个 **Application** 集合。它可以生成 **Application CRD** 资源，进而自动创建多个 **ArgoCD** 应用。不同应用实际上就对应了不同的环境。

那么，**ApplicationSet** 怎么知道要创建几个 **Application** 对象呢？这就需要用到 **ApplicationSet Generators** 了。

ApplicationSet Generators 是一个可以自动生成 **Application** 对象的生成器，它可以通过遍历 **Git** 仓库中的目录来决定生成几个 **Application** 对象。这么说有点抽象，你可以结合下面这张图来进一步理解。



假设我们现在有一个 Helm 应用的 Git 仓库，env 目录下存放了不同环境的 values.yaml 配置文件，那么，ApplicationSet Generators 就可以遍历这些目录，并且自动创建不同环境的 Application 对象，这样就实现了目录和环境的映射关系。也就是说，当我们需要创建一个新的环境时，只需要创建一个目录以及配置文件 values.yaml 就可以了！

这样，不管是为同一个应用创建不同的环境，还是为不同的开发者创建隔离的开发环境，都可以把创建环境等同于创建目录，实现了“代码即环境”。

自动多环境管理实战

在实际创建 ApplicationSet 对象之前，我们先来看一下 [示例仓库](#)。你需要注意的是，这节课所需要用到的示例应用在 helm-env 目录下。

示例应用简介

在将示例应用克隆到本地之后，进入 helm-env 目录，它的目录结构是下面这样的。

```
1 .
2 |— Chart.yaml
3 |— applicationset.yaml
```

```
4 |— env
5 |   |— dev
6 |   |   |— values.yaml
7 |   |— prod
8 |   |   |— values.yaml
9 |   |— test
10 |   |   |— values.yaml
11 |— templates
12 |   |— frontend.yaml
13 |   |— ingress.yaml
```

从它的目录结构可以看出，它由 `Chart.yaml`、`applicationset.yaml`、`env` 目录和 `templates` 目录组成，熟悉 Helm 的同学应该一眼就能看出，其实它是一个 Helm Chart。不同的是，Helm 的配置文件 `values.yaml` 并没有放在 Chart 的根目录，而是放在了 `env` 目录下。

`templates` 目录存放着示例应用的 Kubernetes 对象，为了简化演示过程，这节课我们只部署前端相关的对象，也就是 `frontend.yaml`。

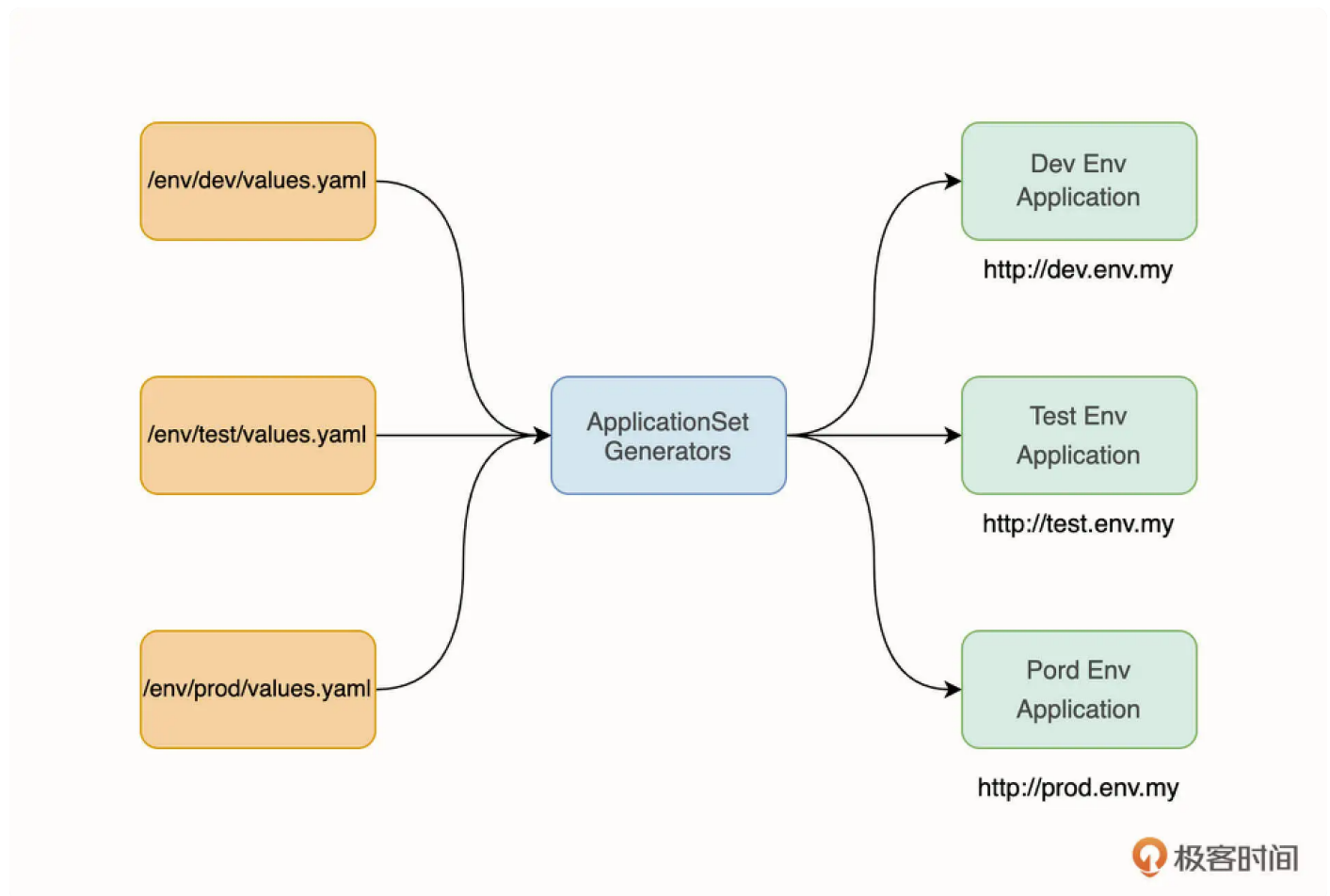
此外，在这个示例应用中，`ingress.yaml` 会用来部署 Ingress 对象，它的内容如下。

 复制代码

```
1 apiVersion: networking.k8s.io/v1
2 kind: Ingress
3 metadata:
4   name: frontend
5   annotations:
6     kubernetes.io/ingress.class: nginx
7 spec:
8   rules:
9     - host: {{ .Release.Namespace }}.env.my
10       http:
11         paths:
12           - path: /
13             pathType: Prefix
14             backend:
15               service:
16                 name: frontend-service
17                 port:
18                   number: 3000
```

需要注意的是，我在 Ingress 对象中使用了 Helm 的内置变量，也就是 `Release.Namespace`，它实际上指的是 Helm Chart 部署的命名空间，我把它和域名做了拼

接。在这节课的例子中，不同的环境将会被部署到独立的命名空间下，这样也就使不同的环境具备了独立的访问域名，如下图所示。



ApplicationSet 简介

我们再来看下 ApplicationSet。ApplicationSet 是这节课介绍的重点，它可以自动生成多个 Application 对象，不同的 Application 对象实际上对应了不同的环境。

示例应用目录下有一个名为 applicationset.yaml 的文件，它定义了 ApplicationSet 的内容。

复制代码

```
1 apiVersion: argoproj.io/v1alpha1
2 kind: ApplicationSet
3 metadata:
4   name: frontend
5   namespace: argocd
6 spec:
7   generators:
8   - git:
9     repoURL: "https://github.com/lyzhang1999/kubernetes-example.git"
10    revision: HEAD
11    files:
```

```
12     - path: "helm-env/env/*/values.yaml"
13   template:
14     metadata:
15       name: "{{path.basename}}"
16     spec:
17       project: default
18       source:
19         repoURL: "https://github.com/lyzhang1999/kubernetes-example.git"
20         targetRevision: HEAD
21         path: "helm-env"
22         helm:
23           valueFiles:
24             - "env/{{path.basename}}/values.yaml"
25       destination:
26         server: 'https://kubernetes.default.svc'
27         namespace: '{{path.basename}}'
28       syncPolicy:
29         automated: {}
30         syncOptions:
31           - CreateNamespace=true
```

这里我们分成两部分来介绍，第一部分是 `spec.generators`，第二部分是 `spec.template`。

Generators 指的是生成器。这里，我们使用 **Git** 生成器，并指定了 **Helm Chart** 仓库地址。请注意，**你需要将这个地址替换为自己的仓库地址**，如果仓库为私有权限，那么还需要在 **ArgoCD** 控制台配置仓库的凭据信息，具体你可以参考 [第 22 讲](#) 的内容。

`revision` 的值的 **HEAD**，指的是远端最新修改的版本。`files` 字段是配置的重点，它通过通配符 `"*"` 号来匹配 `env` 目录下的 `values.yaml` 文件，并为 `template` 字段下的 `path` 变量提供值。

接下来我们继续看 `template` 字段。你可以简单地理解为，`template` 实际上就是在为 **Application** 配置模板，结合生成器，它能够动态生成 **Application** 对象。例如，`metadata.name` 字段配置了每一个 **Application** 的名称，在这个例子中，`path.basename` 变量对应三个值，分别是 `env` 下子目录的名称，也就是 `dev`、`test` 和 `prod`。

`source.repoURL` 字段表示 **Helm Chart** 的来源仓库，**你也需要将它替换为你的仓库地址**。

此外，在 `helm.valueFiles` 里同样也用到了这个变量，在这里，我们为不同的环境指定了不同的 `values.yaml`，这样就实现了环境隔离。

最后，`destination.namespace` 字段也使用了变量，它配置了部署应用的命名空间。

最终，在这个例子中，ApplicationSet 会根据目录结构生成三个 **Application** 对象，而 Application 对象又会在不同的命名空间下部署示例应用，它们分别对应 dev、test 和 prod 环境。

部署 ApplicationSet

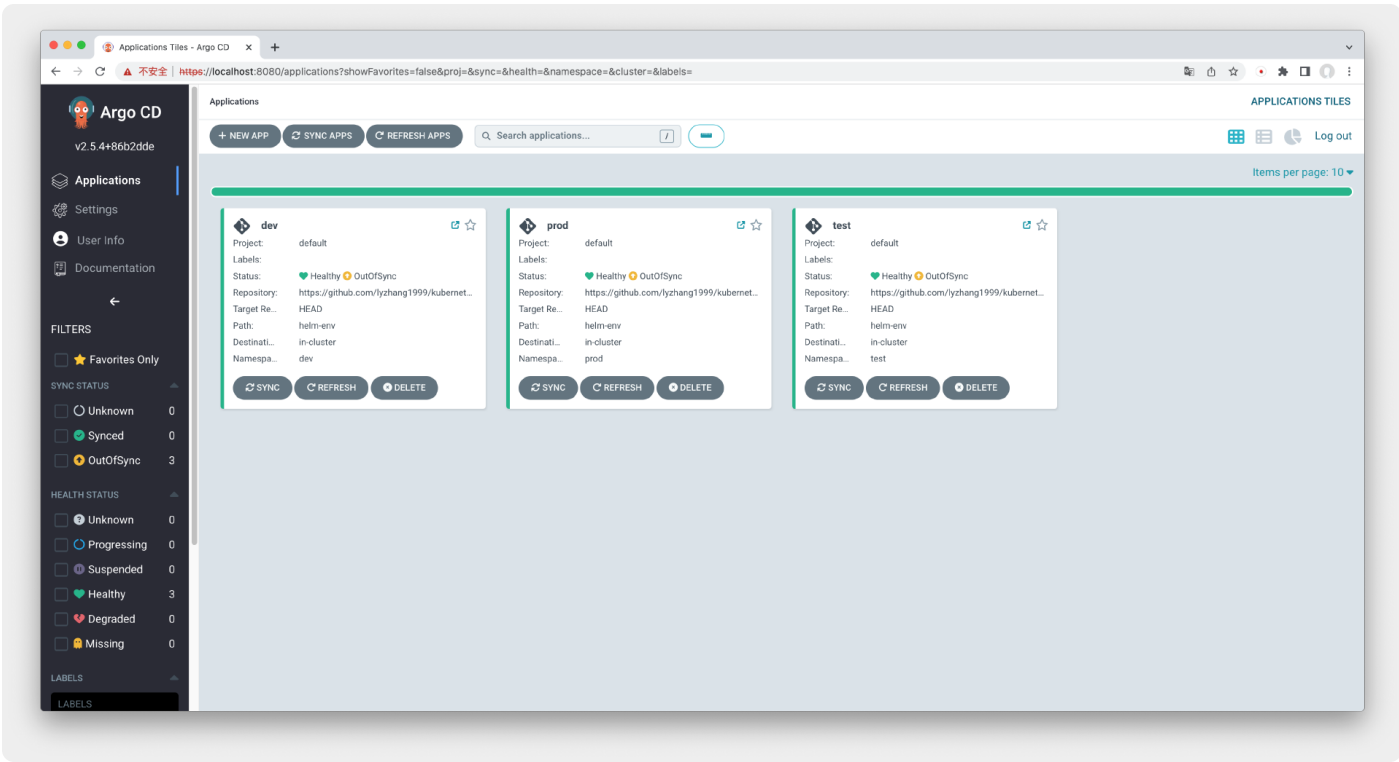
现在，我们尝试部署 ApplicationSet 对象。

你可以使用 `kubectl apply` 命令来部署它。

 复制代码

```
1 $ kubectl apply -f applicationset.yaml
2 applicationset.argoproj.io/frontend created
```

部署完成后，打开 ArgoCD 控制台，你会看到 ApplicationSet 创建了三个应用，名称分别为 dev、test 和 prod，并且它们分别被部署在了不同的命名空间下，如下图所示。



还要注意的，在进入 ArgoCD 控制台之前，你需要进行端口转发操作，并获取 ArgoCD admin 登陆密码，具体操作你可以参考 [第 22 讲](#)。

到这里，ApplicationSet 就已经创建成功了。

访问多环境

接下来，我们尝试访问这三个环境。

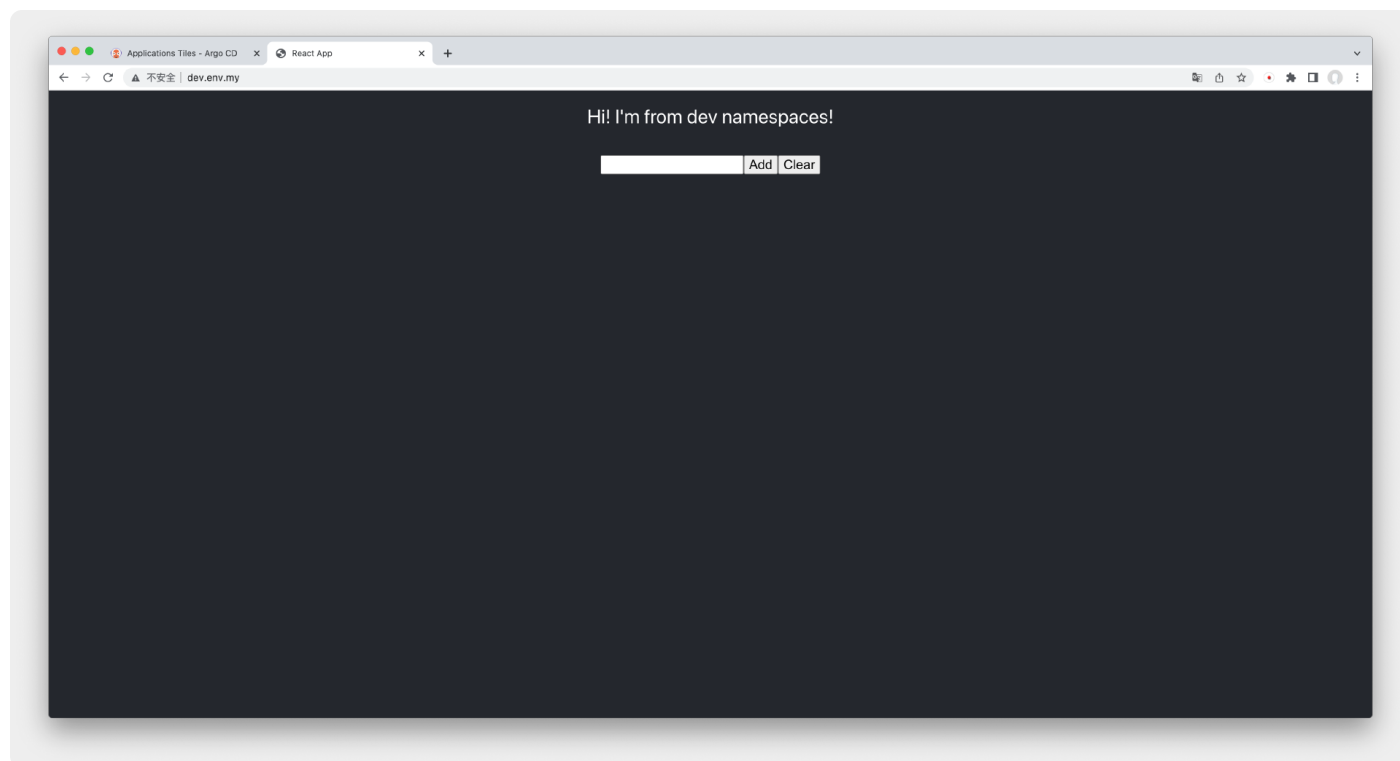
在访问之前，你需要配置下面这三个 Hosts。

复制代码

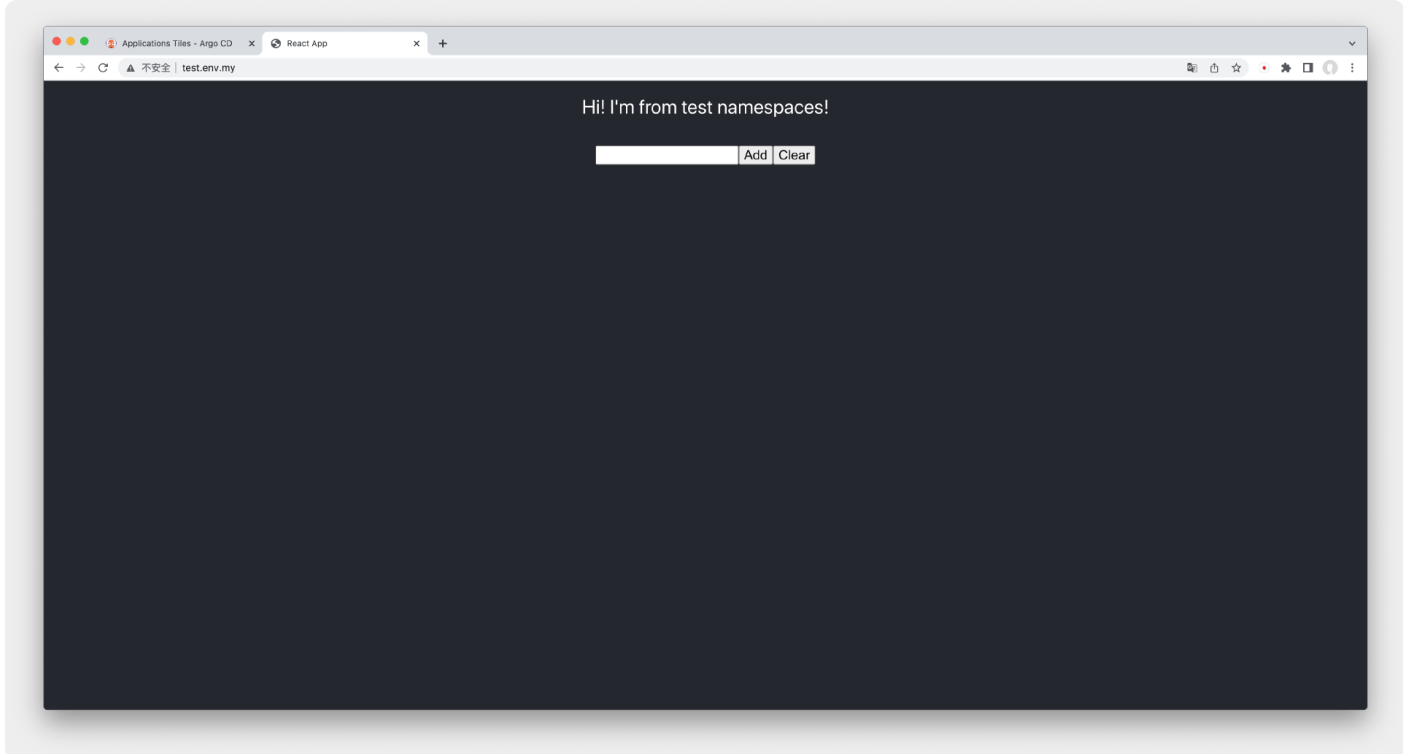
```
1 127.0.0.1 dev.env.my
2 127.0.0.1 test.env.my
3 127.0.0.1 prod.env.my
```

具体的配置方法你可以参考 [第 24 讲](#) 的内容。

Hosts 配置完成后，接下来我们尝试访问**开发环境**。打开浏览器访问 <http://dev.env.my>，你应该能看到下图所示的界面。



然后，你还可以访问**测试环境**。访问链接为 <http://test.env.my>，同样地，你能看到相同的应用界面，只不过返回的内容中命名空间来源产生了变化，如下图所示。



到这里，我们就成功使用 **ApplicationSet** 创建了多个隔离环境。当我们需要对不同的环境进行更新时，只需要更新 **env** 目录下对应环境的 **values.yaml** 文件，就可以触发 **ArgoCD** 自动同步了，不同环境之间互不影响。

此外，当我们需要创建新的环境时，只需要在 **env** 目录下增加一个目录和 **values.yaml** 文件就可以了，**ArgoCD** 会根据配置自动创建新的环境。

创建新环境实验

接下来，我们尝试创建一个新的环境。

首先，在 **env** 目录下创建 **staging** 目录，表示预发布环境。你可以通过下面的命令来创建它。

```
1 cd helm-env/env
2 mkdir staging
```

 复制代码

然后，将 **dev** 目录下的 **values.yaml** 复制到 **staging** 目录下。

```
1 $ cp dev/values.yaml staging
```

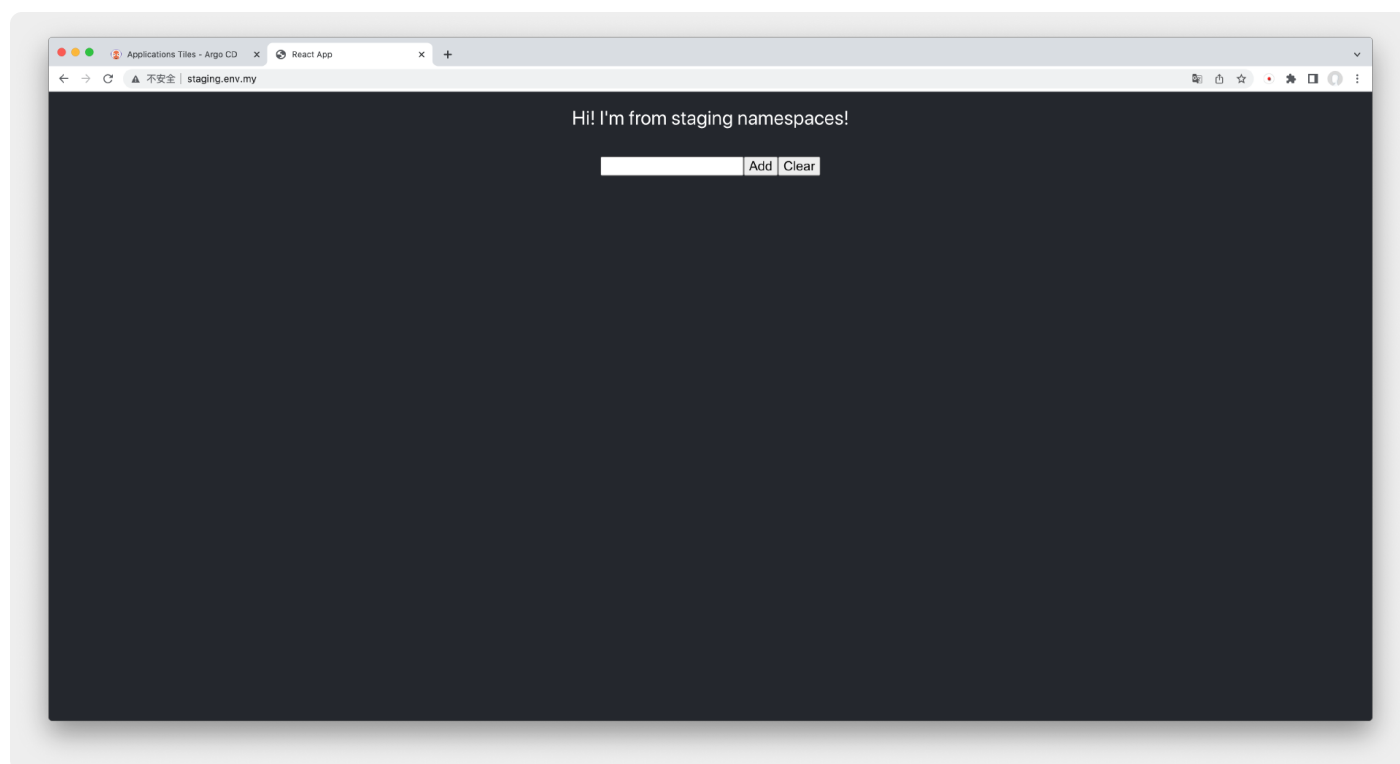
 复制代码

接下来，将修改提交到远端仓库。

 复制代码

```
1 $ git add .  
2 $ git commit -m 'add stagign'  
3 $ git push origin main
```

稍等几分钟，ArgoCD 将自动同步，并为我们创建新的 staging 环境，如下图所示。



到这里，我们就完成了一次创建新环境的全过程。怎么样？创建环境是不是瞬间变得非常简单？

总结

这节课，我们学习了如何通过 **ApplicationSet** 来创建和管理多环境。在实际的业务场景中，我们通常会有多环境的业务需求，相比较传统的创建环境的方式，使用 **ApplicationSet** 大大简化了拉起一个新环境的过程。

“代码即环境”听起来虽然比较抽象，但实现起来并没有这么困难。借助应用定义，结合 **Git** 仓库，我们很容易就可以实现多环境管理。需要注意的是，在众多 **GitOps** 多环境管理的方案中，你可能还会看到另一种在这节课没有介绍的方案：**通过分支来管理多环境**。

我也为你简单对比一下这两种方式。就我的经验来看，采用多分支来管理 GitOps 中的多环境并不能够很好地同时解决可维护性和唯一可信源的问题。首先，分支管理模型会使我们面临差异和合并的问题，这对长期维护来说成本较高，并且在更新环境时，需要切换到不同的分支去操作，这更容易导致人为的错误。其次，分支的管理方式没有目录管理方式来得直观。

所以，在实际的项目中，我推荐你按照这节课的讲解以目录的方式来管理不同的环境。

多环境除了可以用来区分开发环境、测试环境和生产环境之外，还可以很方便地为每一位开发者提供独立的开发环境。

在这节课的例子中，由于所有环境都共用 Helm Chart 的 Template 目录，所以对于应用而言，我们只需要维护 Template 目录就可以间接管理所有的环境了。而对于不同的环境，我们可以使用环境目录下的 values.yaml 文件进行差异化的配置。这样就同时兼顾了可维护性和环境的差异化配置。

思考题


最后，给你留一道思考题吧。

在实际的项目中，我们一般会将环境进行硬隔离，并将它们部署到不同的集群中。请你尝试修改 ApplicationSet，让每个环境在不同的集群中运行。

提示：你需要为 ArgoCD 添加集群，并将 destination 字段配置为集群名称 name 来代替 server，通过目录名和集群名的映射来实现最终目标。

欢迎你给我留言交流讨论，你也可以把这节课分享给更多的朋友一起阅读。我们下节课见。

分享给需要的人，Ta购买本课程，你将得 18 元

 生成海报并分享

 赞 2  提建议

上一篇 26 | 生产稳定的秘密武器：如何实施自动化渐进式交付？

下一篇 28 | 安全提升：GitOps 在哪些环节需要关注安全问题？

更多课程推荐

李三红·搞定 Java 开发基础

极客时间 × 阿里云开发者社区联合出品

李三红
阿里云程序语言
与编译器技术总监
Java Champion



免费订阅 

精选留言 (6)

 写留言



邵涵

2023-03-28 来自北京

思考题，将一套应用定义部署在多个集群中，ApplicationSet定义示例

```
apiVersion: argoproj.io/v1alpha1
```

```
kind: ApplicationSet
```

```
metadata:
```

```
  name: echo-server
```

```
spec:
```

```
  generators:
```

```
    - list:
```

```
      elements:
```

```
        - cluster: production
```

```
          url: https://47.91.XX.XX:6443
```

```
        - cluster: staging
```

```
          url: https://47.111.XX.XX:6443
```

```
template:
```

```
metadata:
  name: '{{cluster}}-echo-server'
spec:
  project: default
  source:
    repoURL: https://code.aliyun.com/shuwei.hsw/echo-server.git
    targetRevision: main
    path: manifests/directory/{{cluster}}
  destination:
    server: '{{url}}'
    namespace: multi-echo-server
```

作者回复： 正确，看来已经掌握了 list generators 类型了，赞！



1



邵涵

2023-03-27 来自北京

几个问题：

1. applicationset.yaml 中可以使用 values.yaml 中定义的变量吗？
2. applicationset.yaml 中定义的 application 模板，是否可以像23讲介绍的那样使用 ArgoCD Image Updater？换句话说，能在发现符合allow-tags过滤条件的新版本的镜像之后，更新工作负载并回写各个环境的values.yaml吗？
3. 如果上边两个问题都是“可以”的话，个人认为应该是可以实现“dev、test、prod环境分别使用从不同的源码分支构建的镜像”这种需求的，但如果上边两个问题是“不可以”的话，要如何实现这个需求呢？为每个环境创建独立的git仓库保存该环境对应的应用定义，进而为每个环境单独创建Application对象，而不使用ApplicationSet？

作者回复：非常好的问题。

两个问题都能实现，applicationset 可以为 Deployment 工作负载定义 Annotations 字段，你只需要把 Image updater 相关的字段定义好就能满足。



2023-03-20 来自北京

这种方式感觉比较适合独立开发测试的服务，如果有比较多关联的上下游服务的话，感觉就会直接拉起一套很庞大的环境，这种情况是不是就不适合这么搞了。

作者回复：几十个微服务其实也还好，结合合理的 resource request 和 limit 可以实现资源超卖。不过对于大型的服务来说，共用一套基础服务的方式也是一个不错的方案。



ghostwritten

2023-03-06 来自北京

总结：

Git & k8s & argocd & ApplationSet

1. 安装 kind
2. 安装argo
3. 安装 ingress
4. 下载 kubernetes-example
5. 明确helm-env名录
6. 定义 template 的ingress.yaml、fronted.yaml变量，例如：命名空间、镜像名称版本
7. 重点理解applationset.yaml的参数定义

扩展：<https://argo-cd.readthedocs.io/en/stable/>

Argocd ApplationSet有以下Generator：

- List Generator （实现多集群）
- Cluster Generator
- Git Generator
- Matrix generator

玩法多多

问题：

ArgoCD + ApplationSet VS ArgoCD +kustomize 多环境（多集群多空间）谁更出色？



po

2023-02-26 来自上海

首先，「不用分支」会使我们面临差异和合并的问题，这对长期维护来说成本较高，并且在更新环境时，需要切换到不同的分支去操作，这更容易导致人为的错误。其次，分支的管理方式没有目录管理方式来得直观。

=====

看上下文内容，这里应该是 不用分支 --> 用分支？

作者回复：是的，感谢指正。



橙汁

2023-02-19 来自北京

有种实际场景借助applicationset在不同集群中创建了各自的环境，接下来想进行镜像更新想到两种实现方案。

1. 借助images updater插件，根据23章内容来看是修改application资源，当前application是自动创建不确定修改后是否会有问题。
 2. 将打包成功后镜像的tag更新到helm仓库中，也就是value.yaml文件。假设代码和部署是两个仓库，就需要在ci最后阶段更新helm仓库的value.yaml。
- 希望老师能指正下思路是否正确。

作者回复: 第二种方案是可行的。

第一种方案值得实践一下。

共 2 条评论 >

