Kustomize

Declarative

You can define a Kustomize application manifest in the declarative GitOps way. Here is an example:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: kustomize-example
spec:
  project: default
  source:
    path: examples/helloWorld
    repoURL: 'https://github.com/kubernetes-sigs/kustomize'
    targetRevision: HEAD
  destination:
    namespace: default
    server: 'https://kubernetes.default.svc'
```

If the `kustomization.yaml` file exists at the location pointed to by `repoURL` and `path`, Argo CD will render the manifests using Kustomize.

The following configuration options are available for Kustomize:

- `namePrefix` overrides the namePrefix in the kustomization.yaml for Kustomize apps
- `nameSuffix` overrides the nameSuffix in the kustomization.yaml for Kustomize apps
- `images` is a list of Kustomize image overrides
- `replicas` is a list of Kustomize replica overrides
- `commonLabels` is a string map of additional labels
- `labelWithoutSelector` is a boolean value which defines if the common label(s) should be applied to resource selectors. It also excludes common labels from templates unless `labelIncludeTemplates` is set to true.
- `labelIncludeTemplates` is a boolean value which defines if the common label(s) should be applied to resource templates.
- `commonAnnotations` is a string map of additional annotations

stable

- `namespace` is a Kubernetes resources namespace
- `commonAnnotationsEnvsubst` is a boolean value which enables env variables substitution in annotation values
- `patches` is a list of Kustomize patches that supports inline updates
- `components` is a list of Kustomize components
- `ignoreMissingComponents` prevents kustomize from failing when components do not exist locally by not appending them to kustomization file
- `forceCommonLabels` is a boolean value. When true, Argo CD passes --force to kustomize edit add label, allowing an existing commonLabels/labels entry in kustomization.yaml to be replaced. When false, generation fails if the label key already exists.
- `forceCommonAnnotations` is a boolean value. When true, Argo CD passes --force to kustomize edit add annotation, allowing an existing commonAnnotations entry in kustomization.yaml to be replaced. When false, generation fails if the annotation key already exists.

To use Kustomize with an overlay, point your path to the overlay.

> **Tip**
>
> If you're generating resources, you should read up how to ignore those generated resources using the `IgnoreExtraneous` compare option.

Patches

Patches are a way to kustomize resources using inline configurations in Argo CD applications. `patches` follow the same logic as the corresponding Kustomization. Any patches that target existing Kustomization file will be merged.

This Kustomize example sources manifests from the `/kustomize-guestbook` folder of the `argoproj/argocd-example-apps` repository, and patches the `Deployment` to use port `443` on the container.

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
metadata:
  name: kustomize-inline-example
namespace: test1
resources:
```

```
    - https://github.com/argoproj/argocd-example-apps//kustomize-guestbook/
  patches:
    - target:
        kind: Deployment
        name: guestbook-ui
      patch: |-
        - op: replace
          path: /spec/template/spec/containers/0/ports/0/containerPort
          value: 443
```

This `Application` does the equivalent using the inline `kustomize.patches` configuration.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: kustomize-inline-guestbook
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  destination:
    namespace: test1
    server: https://kubernetes.default.svc
  project: default
  source:
    path: kustomize-guestbook
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: master
    kustomize:
      patches:
        - target:
            kind: Deployment
            name: guestbook-ui
          patch: |-
            - op: replace
              path: /spec/template/spec/containers/0/ports/0/containerPort
              value: 443
```

The inline kustomize patches work well with `ApplicationSets`, too. Instead of maintaining a patch or overlay for each cluster, patches can now be done in the `Application` template and utilize attributes from the generators. For example, with `external-dns` to set the `txt-owner-id` to the cluster name.

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: external-dns
spec:
  goTemplate: true
  goTemplateOptions: ["missingkey=error"]
```

stable

```
  generators:
  - clusters: {}
  template:
    metadata:
      name: 'external-dns'
    spec:
      project: default
      source:
        repoURL: https://github.com/kubernetes-sigs/external-dns/
        targetRevision: v0.14.0
        path: kustomize
        kustomize:
          patches:
          - target:
              kind: Deployment
              name: external-dns
            patch: |-
              - op: add
                path: /spec/template/spec/containers/0/args/3
                value: --txt-owner-id={{.name}}   # patch using attribute from
generator
      destination:
        name: 'in-cluster'
        namespace: default
```

Components

Kustomize components encapsulate both resources and patches together. They provide a powerful way to modularize and reuse configuration in Kubernetes applications. If Kustomize is passed a non-existing component directory, it will error out. Missing component directories can be ignored (meaning, not passed to Kustomize) using `ignoreMissingComponents`. This can be particularly helpful to implement a [default/override pattern].

Outside of Argo CD, to utilize components, you must add the following to the `kustomization.yaml` that the Application references. For example:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
...
components:
- ../component
```

With support added for components in `v2.10.0`, you can now refer

stable

directly in the Application:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: application-kustomize-components
spec:
  ...
  source:
    path: examples/application-kustomize-components/base
    repoURL: https://github.com/my-user/my-repo
    targetRevision: main

    # This!
    kustomize:
      components:
        - ../component  # relative to the kustomization.yaml (`source.path`).
      ignoreMissingComponents: true
```

Private Remote Bases

If you have remote bases that are either (a) HTTPS and need username/password (b) SSH and need SSH private key, then they'll inherit that from the app's repo.

This will work if the remote bases use the same credentials/private key. It will not work if they use different ones. For security reasons your app only ever knows about its own repo (not other team's or users repos), and so you won't be able to access other private repos, even if Argo CD knows about them.

Read more about private repos.

`kustomize build` Options/Parameters

To provide build options to `kustomize build` of default Kustomize version, use `kustomize.buildOptions` field of `argocd-cm` ConfigMap. Use `kustomize.buildOptions.<version>` to register version specific build options.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
    app.kubernetes.io/name: argocd-cm
    app.kubernetes.io/part-of: argocd
data:
```

stable

```
    kustomize.buildOptions: --load-restrictor LoadRestrictionsNone
    kustomize.buildOptions.v4.4.0: --output /tmp
```

After modifying `kustomize.buildOptions`, you may need to restart ArgoCD for the changes to take effect.

Custom Kustomize versions

Argo CD supports using multiple Kustomize versions simultaneously and specifies required version per application. To add additional versions make sure required versions are bundled and then use `kustomize.path.<version>` fields of `argocd-cm` ConfigMap to register bundled additional versions.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
    app.kubernetes.io/name: argocd-cm
    app.kubernetes.io/part-of: argocd
data:
    kustomize.path.v3.5.1: /custom-tools/kustomize_3_5_1
    kustomize.path.v3.5.4: /custom-tools/kustomize_3_5_4
```

Once a new version is configured you can reference it in an Application spec as follows:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
spec:
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: kustomize-guestbook

    kustomize:
      version: v3.5.4
```

Additionally, the application kustomize version can be configured using the Parameters tab of the Application Details page, or using the following CLI comm

stable

```
argocd app set <appName> --kustomize-version v3.5.4
```

Build Environment

Kustomize apps have access to the standard build environment which can be used in combination with a config management plugin to alter the rendered manifests.

You can use these build environment variables in your Argo CD Application manifests. You can enable this by setting `.spec.source.kustomize.commonAnnotationsEnvsubst` to `true` in your Application manifest.

For example, the following Application manifest will set the `app-source` annotation to the name of the Application:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook-app
  namespace: argocd
spec:
  project: default
  destination:
    namespace: demo
    server: https://kubernetes.default.svc
  source:
    path: kustomize-guestbook
    repoURL: https://github.com/argoproj/argocd-example-apps
    targetRevision: HEAD
    kustomize:
      commonAnnotationsEnvsubst: true
      commonAnnotations:
        app-source: ${ARGOCD_APP_NAME}
  syncPolicy:
    syncOptions:
      - CreateNamespace=true
```

Kustomizing Helm charts

It's possible to render Helm charts with Kustomize. Doing so requires that you pass the `--enable-helm` flag to the `kustomize build` command. This flag is not part of the Kustomize options within Argo CD. If you would like to render Helm charts through Kustomize in an Argo CD application, you have two options: You can either create a custom plugin, or modify the `argocd-cm` ConfigMap to include the `--enable-helm` flag globally for all Kustomize applications:

stable

```yaml
apiVersion: v1
kind: ConfigMap
```

```yaml
metadata:
  name: argocd-cm
  namespace: argocd
data:
  kustomize.buildOptions: --enable-helm
```

Setting the manifests' namespace

The `spec.destination.namespace` field only adds a namespace when it's missing from the manifests generated by Kustomize. It also uses `kubectl` to set the namespace, which sometimes misses namespace fields in certain resources (for example, custom resources). In these cases, you might get an error like this: `ClusterRoleBinding.rbac.authorization.k8s.io "example" is invalid: subjects[0].namespace: Required value.`

Using Kustomize directly to set the missing namespaces can resolve this problem. Setting `spec.source.kustomize.namespace` instructs Kustomize to set namespace fields to the given value.

If `spec.destination.namespace` and `spec.source.kustomize.namespace` are both set, Argo CD will defer to the latter, the namespace value set by Kustomize.

stable