Octopus Deploy

GitOps and Secrets

ArgoCon EU 2026

Kostis Kapelonis

Kostis Kapelonis

Developer Advocate (Octopus Deploy)

Member of Argo CD, Argo Rollouts

Co-author GitOps certification
http://learning.octopus.com

Agenda

1. Intro
2. Demo (ESO + Vault + Argo CD)
3. Choices for secret management
4. Your applications
5. Their applications
6. Conclusion

The problem: secrets and GitOps

Question 1 - How do I install Argo CD?

Question 2 - How can secrets work with GitOps?

- GitOps principes say that I should store **everything** in Git
- Storing raw secrets in Git is a recipe for disaster.

Argo CD and secrets

100,000 Different Ways to Manage Secrets in GitOps - Andrew Block, Red Hat

Secrets Management the Argo CD Way | Argo Unpacked #4

Photo by Eila Lifflander on Unsplash

Before and after Argo CD 3.0

Secret Management

Argo CD is un-opinionated about how secrets are managed. There are many ways to do it, and there's no one-size-fits-all solution.

Many solutions use plugins to inject secrets into the application manifests. See Mitigating Risks of Secret-Injection Plugins below to make sure you use those plugins securely.

Here are some ways people are doing GitOps secrets:

- Bitnami Sealed Secrets
- External Secrets Operator
- Hashicorp Vault
- Banzai Cloud Bank-Vaults
- Helm Secrets
- Kustomize secret generator plugins
- aws-secret-operator
- KSOPS
- argocd-vault-plugin
- argocd-vault-replacer
- Kubernetes Secrets Store CSI Driver

Just a list of many different secret solutions

For discussion, see #1364

Destination Cluster Secret Management

In this approach, secrets are populated on the destination cluster, and Argo CD does not need to directly manage them. Sealed Secrets, External Secrets Operator, and Kubernetes Secrets Store CSI Driver are examples of this style of secret management.

This approach has two main advantages:

1) Security: Argo CD does not need to have access to the secrets, which reduces the risk of leaking them. 2) User Experience: Secret updates are decoupled from app sync operations, which reduces the risk of unintentionally applying Secret updates during an unrelated release.

We strongly recommend this style of secret management.

Other examples of this style of secret management include: * aws-secret-operator * Vault Secrets Operator

What we recommend

Argo CD Manifest Generation-Based Secret Management

In this approach, Argo CD's manifest generation step is used to inject secrets. This may be done using a Config Management Plugin like argocd-vault-plugin.

We strongly caution against this style of secret management, as it has several disadvantages:

1) Security: Argo CD needs access to the secrets, which increases the risk of leaking them. Argo CD stores generated manifests in plaintext in its Redis cache, so injecting secrets into the manifests increases risk. 2) User Experience: Secret updates are coupled with app sync operations, which increases the risk of unintentionally applying Secret updates during an unrelated release. 3) Rendered Manifests Pattern: This approach is incompatible with the "Rendered Manifests" pattern, which is increasingly becoming a best practice for GitOps.

Many users have already adopted generation-based solutions, and we understand that migrating to an operator-based solution can be a significant effort. Argo CD will continue to support generation-based secret management, but we will not prioritize new features or improvements that solely support this style of secret management.

Avoid this

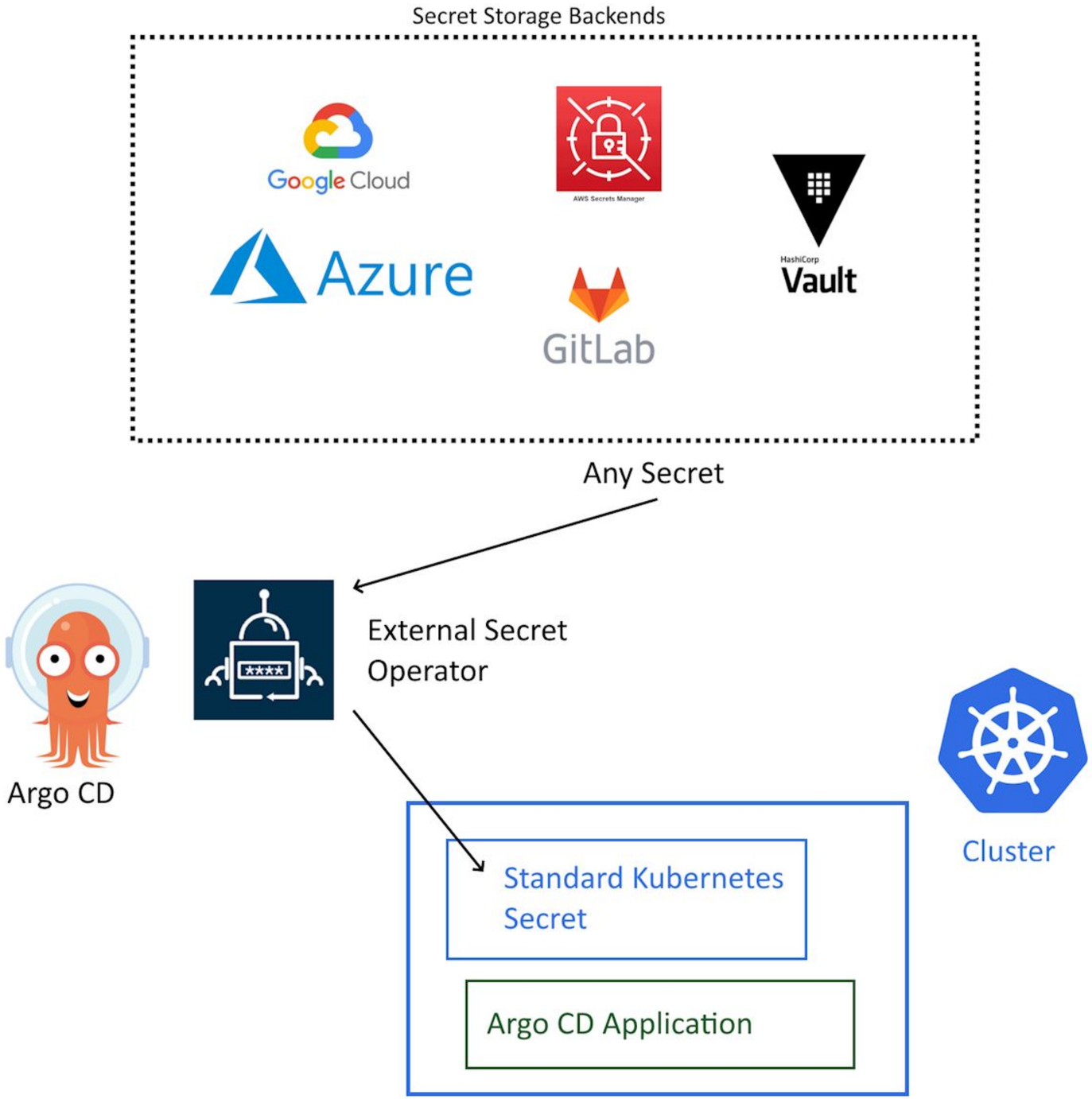
https://secrets-store-csi-driver.sigs.k8s.io/

TL;DR

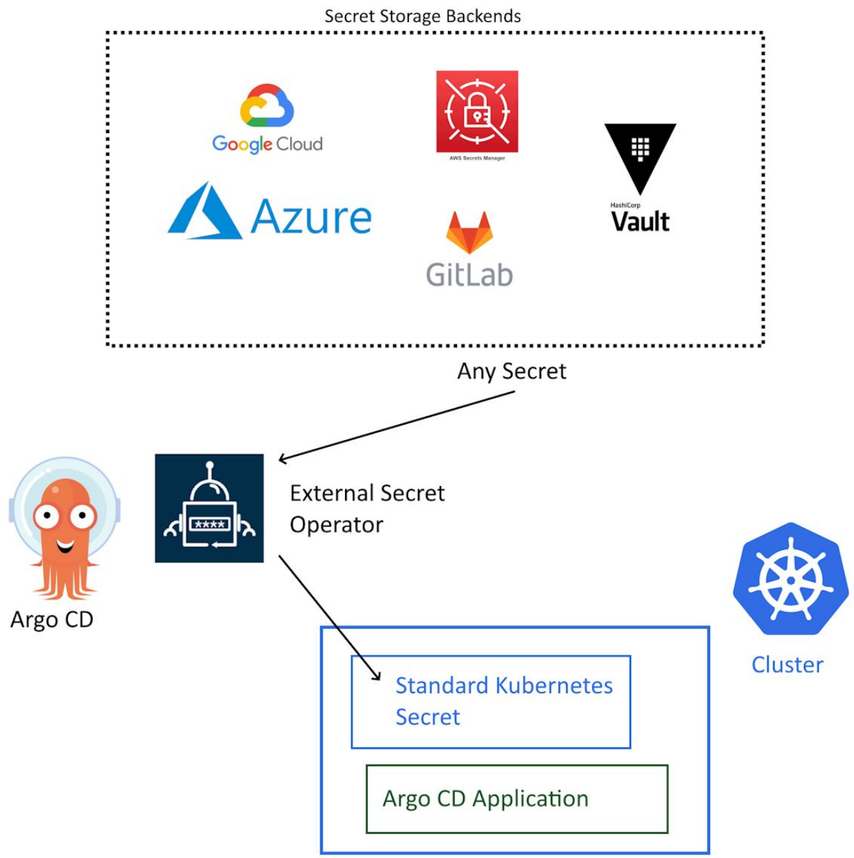
Use External Secret Operator or the SS CSI driver

https://external-secrets.io

Demo（External Secrets)



- **Vault**
- **ESO**
- **Argo CD**

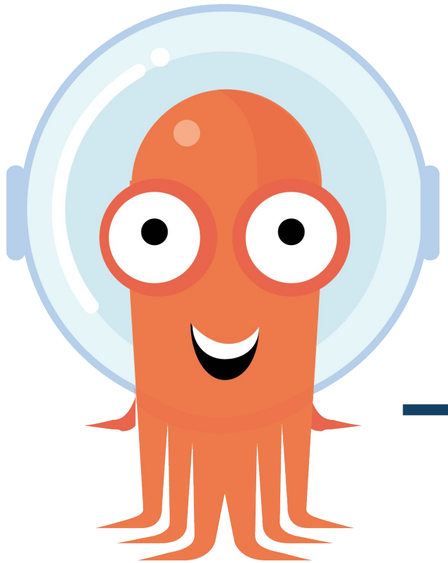


https://github.com/kostis-codefresh/external-secrets-gitops-example

If changing a secret requires a sync operation, you are doing it wrong

If your secret solution is tightly coupled to Argo CD you are doing it wrong

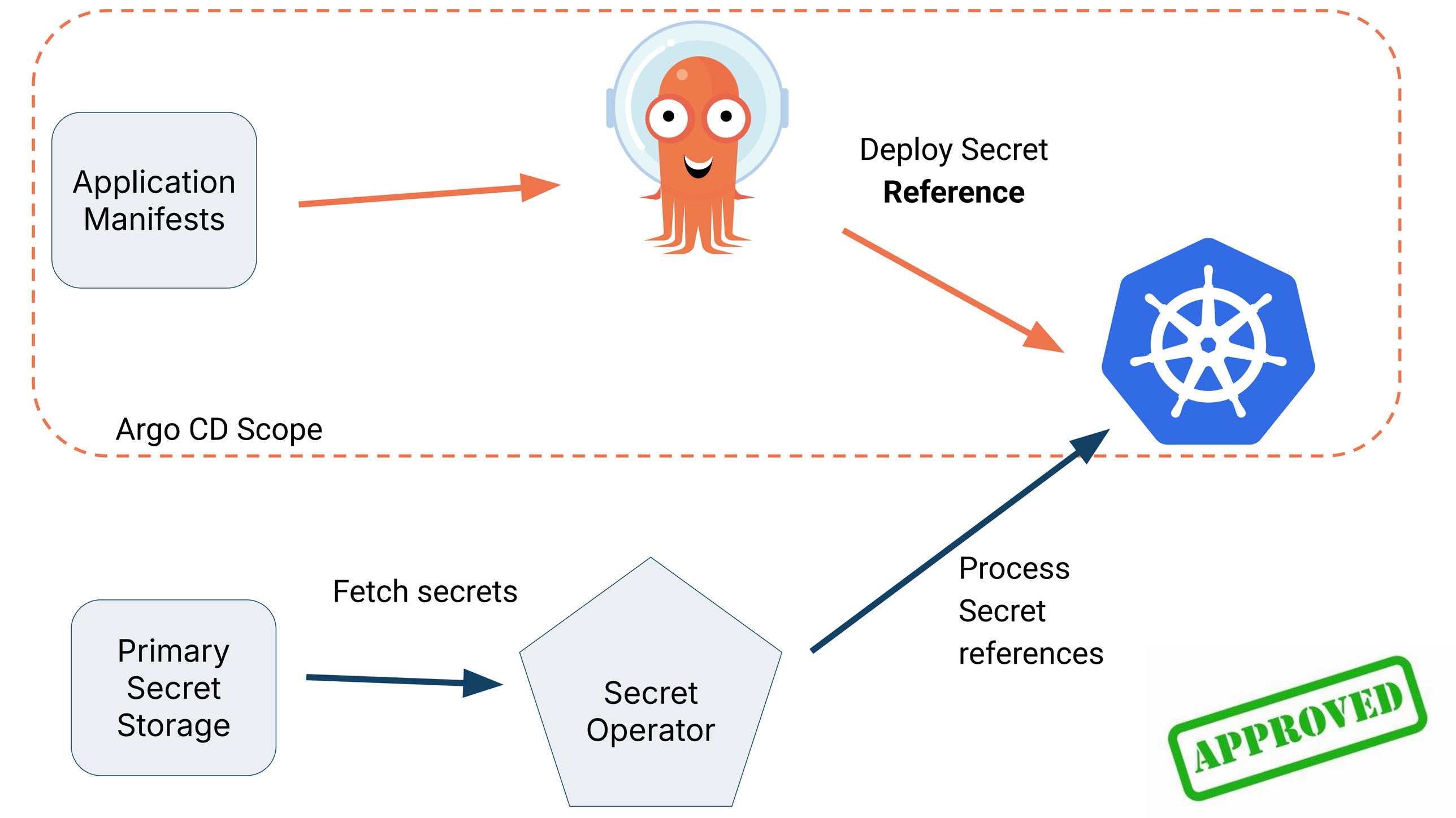
Argo CD is NOT a secret management solution



Inject or process or template or secrets

Fetch secrets

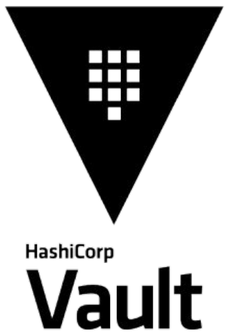
Primary Secret Storage


Application Manifests
Deploy Secret
Reference
Argo CD Scope
Primary Secret Storage
Fetch secrets
Secret Operator
Process Secret references
APPROVED

Application Manifests

Deploy Secret
Reference

Argo CD Scope



Fetch secrets

Process
Secret
references

Argo CD handles a REFERENCE to a secret

...and not the secret value itself....

Argo CD deploys

```yaml
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: my-db-credentials
spec:
  refreshInterval: "15s"
  secretStoreRef:
    name: vault-backend
    kind: ClusterSecretStore
  target:
    name: mysql-credentials
```

ESO converts it to actual secret

How to choose a secret solution

Want “Pure GitOps” Solution → Use Bitnami Sealed Secrets

Want to make your life easy → **Use the External Secret Operator** 

Don’t want K8s secrets at all → Use the Secret Store CSI Driver

Just use External Secrets

https://external-secrets.io

Your applications

Do NOT use Environment variables for secrets

You can’t change an env variable after a process has started.

Make your applications smarter

1. The application should read conf from **files**
2. DB/Queue URL must be configurable
3. Settle on conventions (e.g. /secrets)
4. Application should **auto-reload** conf on its own
5. You need to coordinate with your developers for this

Work WITH your Developers

- Photo by Sylvain Mauroux on Unsplash

Popular languages support

- Viper Conf (Golang)
- RefreshScope (Spring/Java)
- chokidar/config (Node.js)
- configparser/watchdog (Python)
- yaml/listen (Ruby)
- config/watchservice (Kotlin)
- config/config-watch (Rust)
- symfony/config-filesystem (PHP)

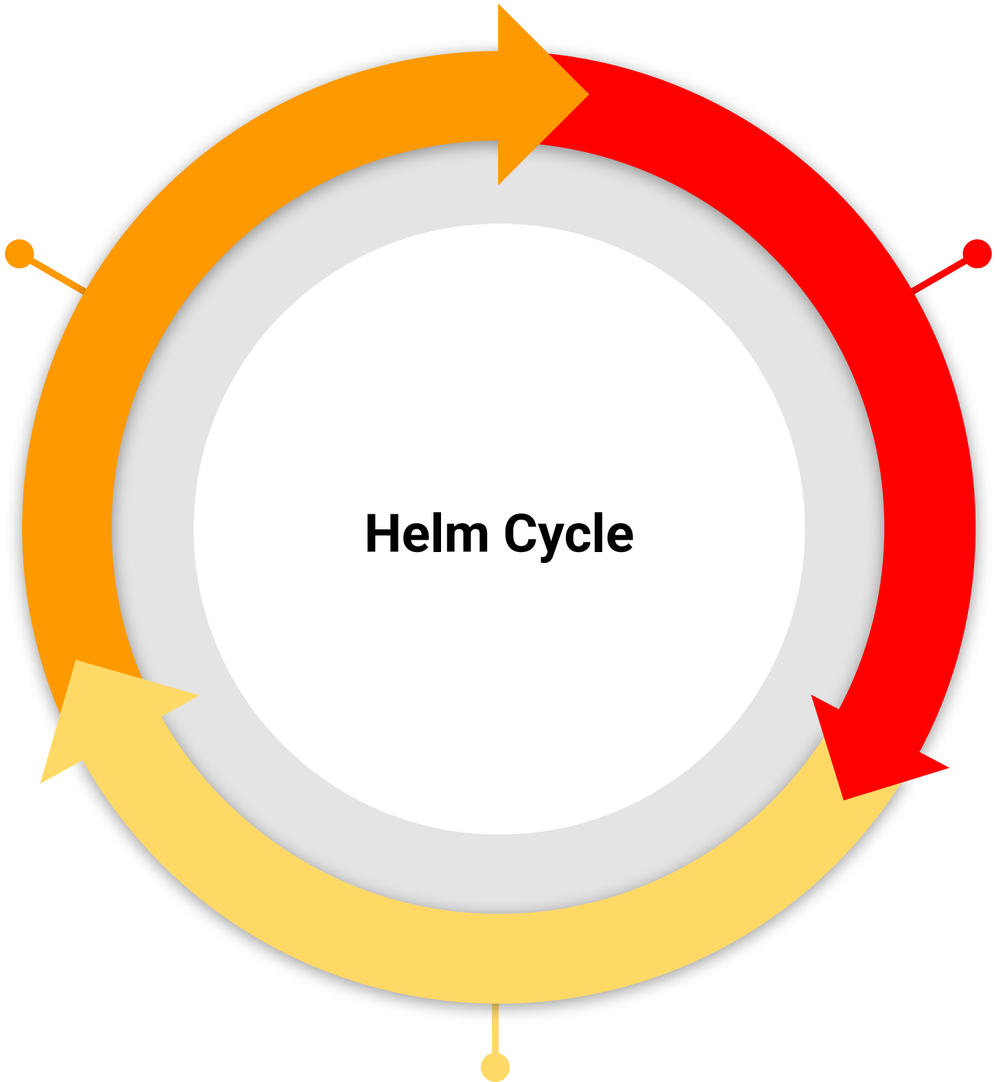
Secret Rotation made easy (before and after)

1. Panic !
2. Find which applications are affected
3. Change the secrets
4. Resync apps
5. Miss some pods
6. Kill all pods to reload their secrets
7. Hope that you rotated everything

1. Rotate the secret
2. Done!

Their applications

People create charts with hardcoded secrets in values



Hacks/ Workarounds/Plugins for Argo CD to load secrets

People can use badly designed charts with Argo CD

Break the cycle! Fix your charts

```yaml
spec:
  template:
    spec:
      containers:
        - name: app
          image: my-app:latest
          env:
            - name: DB_PASSWORD
              value: {{ .Values.database.password | quote }}
```


```yaml
spec:
  template:
    spec:
      containers:
        - name: app
          image: my-app:latest
          env:
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: {{ .Values.database.existingSecretName }}
                  key: {{ .Values.database.existingSecretKey }}
```

Thanks for the assist!

Chart should allow specifying existing secrets #189

Closed

josecv opened on Mar 15, 2019 · edited by josecv — Edits

Right now, this helm chart generates all the secrets, unconditionally, from values in the `values.yaml`. For those of us who like to commit our `values.yaml`s this is not ideal, as it requires that we commit our passwords, for example our S3 access key.

It seems like the best practice is to allow the user to supply an `existingSecret` (e.g. in the postgres chart).

Something similar could be done here -- for example for the registry, if `registry.existingSecret` is set, don't generate a secret and just trust the user to figure out what the secret should look like on their own (which allows them to self-manage the lifecycle of their passwords).

If there's interest, I can find some time to do this PR.

57

Support "existingSecret" for secrets used in jupyterhub chart #1622

Closed

lsowen opened on Apr 10, 2020

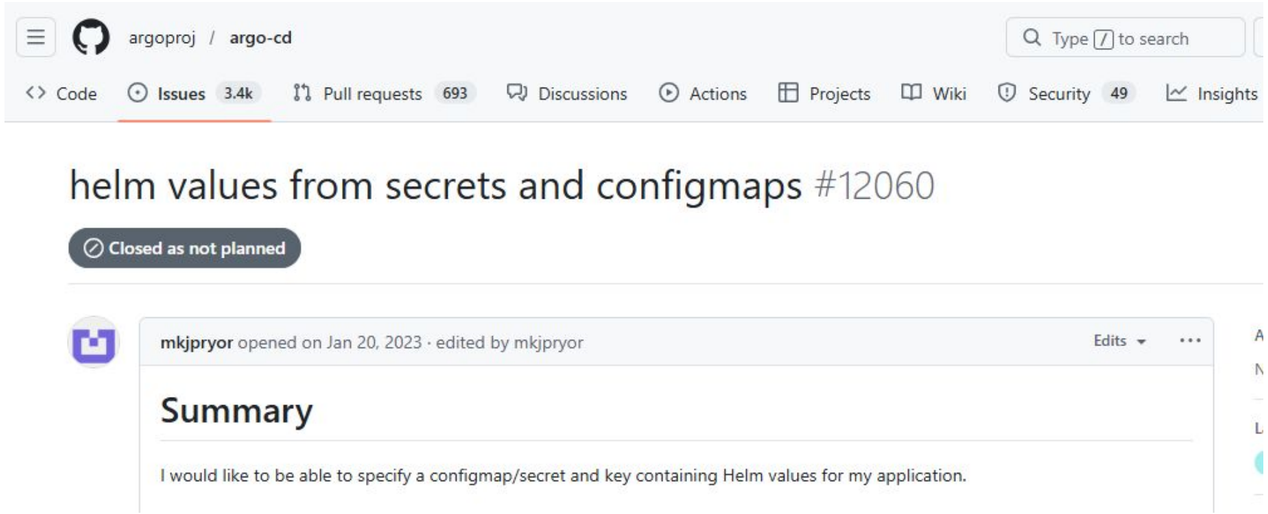
It would be nice to be able to pass a reference to an "existingSecret" to the jupyterhub helm chart, and have the helm chart reference values from that secret instead of requiring secret values (eg `.Values.hub.db.password`) to be specified. This would be in addition to the current process, to eliminate breaking changes and to allow ease of setup/configuration.

An example in a chart which implements this feature is redis:

- Specifying in values.yaml
- Disabling the helm managed secret if an "existingSecret" is specified: secret.yaml
- Supporting referencing helm managed secret or "existingSecret": redis-master-statefulset.yaml

In my specific case, we have a system which manages and deploys secrets separately. This is for security (both to limit the number of people who can access the secrets, and to prevent helm tiller from storing the secret value).

Issues that drag Argo CD to secret management

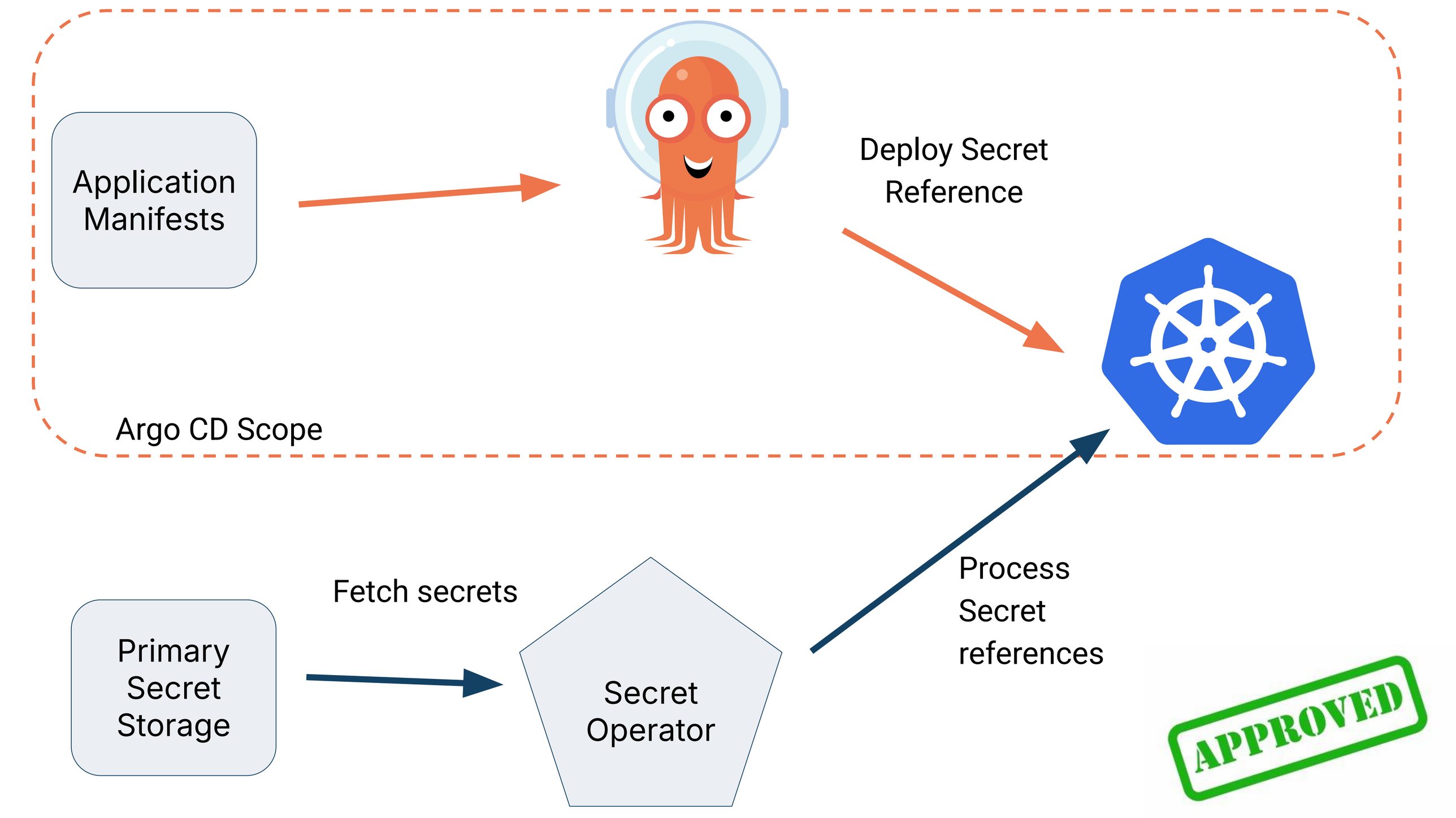


There are several open issues and discussions that if you boil them down they ask for Argo CD to become a secret management solution

Conclusion

Just use External Secrets

https://external-secrets.io


Application Manifests
Deploy Secret Reference
Argo CD Scope
Primary Secret Storage
Fetch secrets
Secret Operator
Process Secret references
APPROVED

Argo CD is NOT a secret management solution

...and it will **NOT** become one in the future...

Please don't open issues/PRs that attempt to convert Argo CD to a secret management solution

Thank you!

Questions: kostis.kapelonis@octopus.com

GitOps/Argo CD certification learning.octopus.com

CNCF Slack https://slack.cncf.io/

Support: https://octopus.com/support/enterprise-argo-support

Octopus Deploy