Octopus Deploy

ARGOCON NORTH AMERICA 2026 · SALT LAKE CITY

Securing the Repo Server

Native mTLS between Argo CD and its repo-server

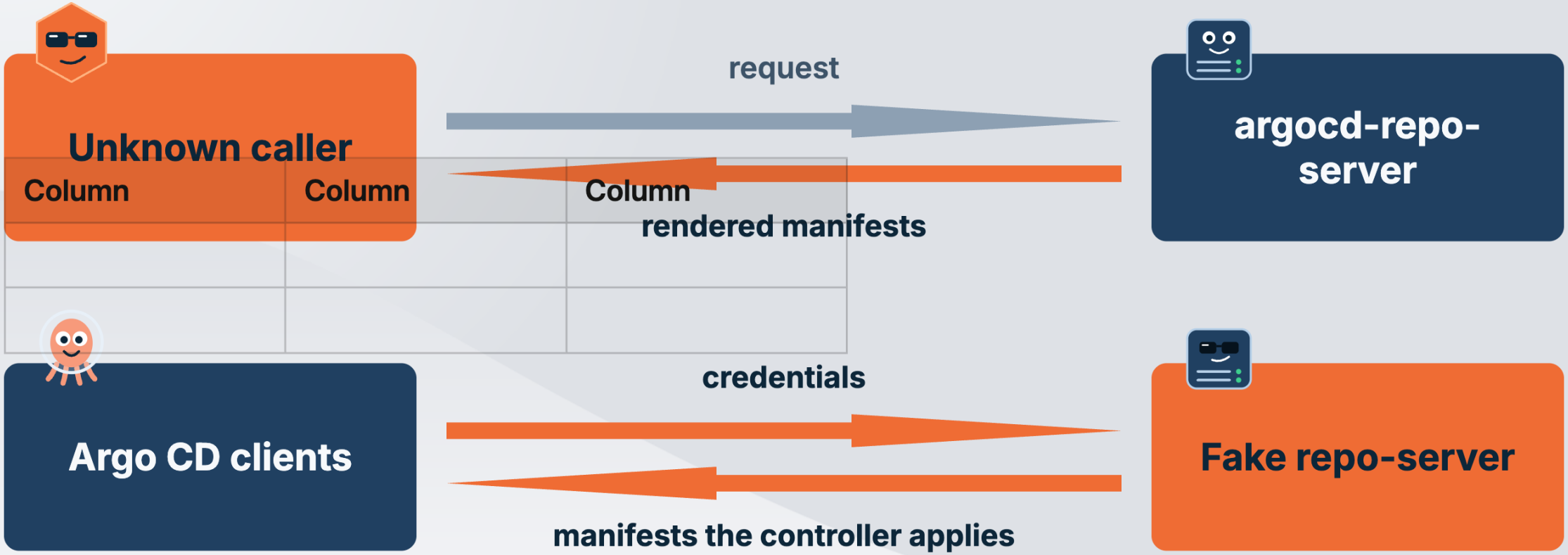


ARGO CD MAINTAINER

Patroklos Papapetrou

Lead Open Source Engineer @ Octopus Deploy

TLS is enabled by default, but there's no certificate validation



DEMO · NO MTLS YET

A pod that can reach repo-server can get rendered manifests

▷ The action

A pod that isn't part of Argo CD asks the repo-server to render the guestbook app, with no client certificate.

</> The repo-server

It returns the manifests, including the message from the guestbook ConfigMap.

The network policy

The default network policy only lets Argo CD's own pods reach this port, and I added a rule for this pod.



THE REPO-SERVER

To enable mTLS, add client-ca.crt and restart

The file

client-ca.crt, the CA that signs your client certificates, in this Secret.

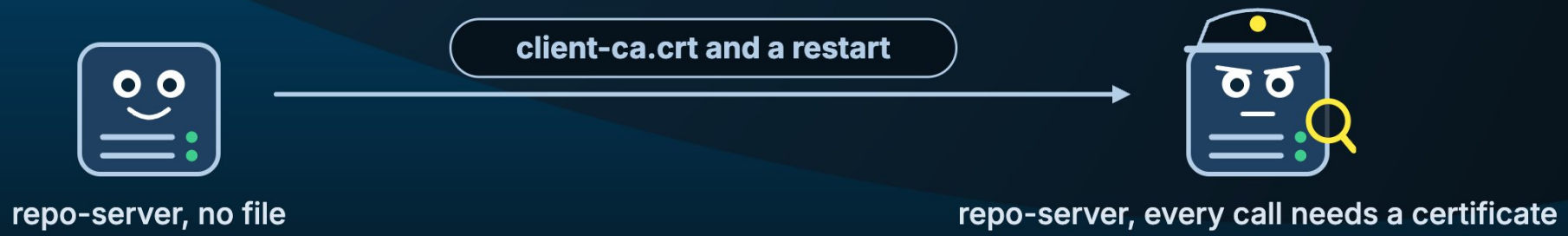
```
argocd-repo-server-mtls
  client-ca.crt
```

At startup

When the file is there, every call needs a certificate from that CA. There are no flags to set.

With no file

The repo-server skips mTLS and reports no error, so check for the startup log line mTLS enabled for repo-server.



DEMO · MTLS ON THE REPO-SERVER

The repo-server refuses the pod and Argo CD's clients

▷ The action

I add client-ca.crt to the Secret and restart the repo-server. The same pod repeats its call, and I sync the app.

</> The repo-server

From now on it requires a client certificate, and it refuses the pod's call at the TLS handshake.

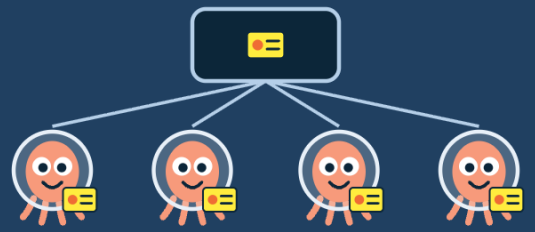
Argo CD

Its clients have no certificate yet, so they are refused as well, and the sync fails until they get one.



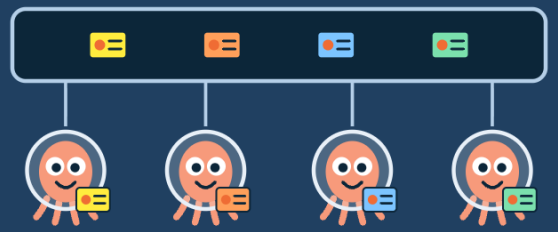
Clients need a certificate: three options

1



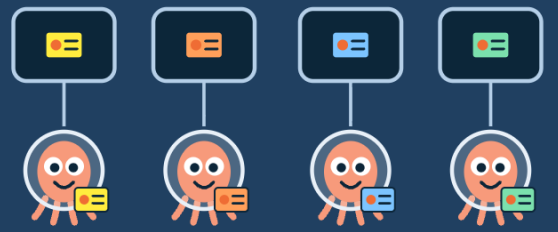
One client certificate shared by all components

2



A certificate per component, all in one Secret

3

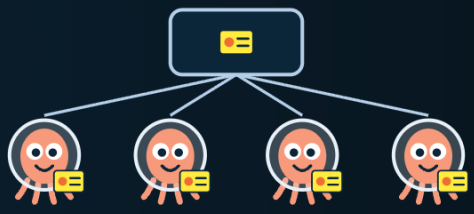


A certificate per component, each in its own Secret



OPTION 1 · SHARED CERTIFICATE

One certificate for every client



How it works

Add client.crt and client.key to the same Secret and restart the four clients. In the stock manifests each one reads them from a volume it already mounts.

```
argocd-repo-server-mtls
  client-ca.crt
  client.crt
  client.key
/app/config/reposerver/mtls
```

Why use this option

No patches on the stock manifests, and the simplest way to make the repo-server check its clients.

The downside

All four clients share one certificate, so if the key is leaked then all communication for these clients is compromised

DEMO · SHARED CERTIFICATE

The clients get the shared certificate and the repo-server accepts their calls

▷ The action

I add client.crt and client.key to the Secret, restart the four clients and sync the app again.

</> The repo-server

It doesn't restart this time, because the new certificate is signed by the CA it already trusts.

Argo CD

Every client sends the shared certificate on its next call, the repo-server accepts it, and the sync works.



DEMO · SHARED CERTIFICATE

The call with no certificate is still refused, and the shared certificate works

▷ The action

The same pod repeats the call with no certificate, then with the shared certificate.

</> The repo-server

It refuses the first call at the TLS handshake and returns manifests for the second. It doesn't log the refusal at the default log level.



THE CONFIGMAP

mTLS is now enabled, but the clients don't check the repo-server yet

What you have now

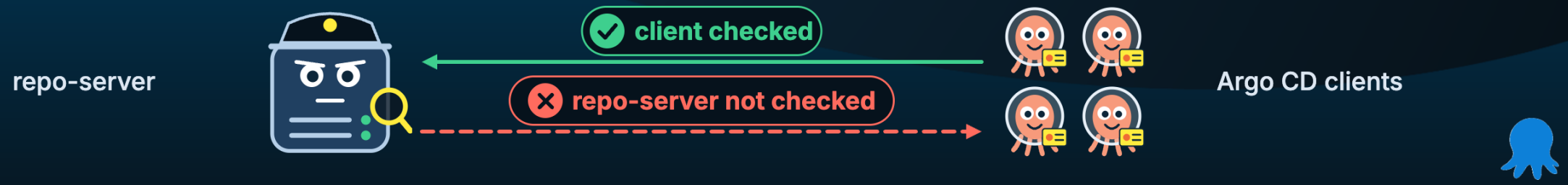
The repo-server checks every client. Argo CD's components read their startup settings from a ConfigMap, argocd-cmd-params-cm, which has nothing related to mTLS.

What's missing

With no CA set, a client accepts any certificate and sends its requests, including any repository credentials, to any server that answers at the repo-server's address.

Action

Give the repo-server a certificate signed by your CA. Then, for each client, set the path to that CA in the ConfigMap, such as controller.repo.server.ca.cert.path.



DEMO · THE CLIENTS VERIFY

The clients now check the repo-server, and the sync still works

▷ The action

I give the repo-server a certificate from our CA, set the CA path for each client, restart them and sync the app.

</> The repo-server

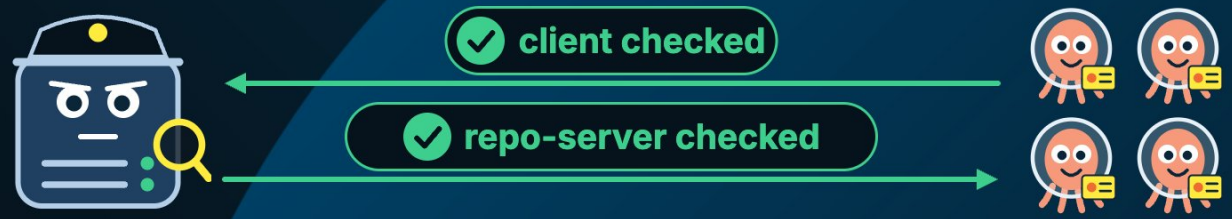
The manual's Secret gives all four clients tls.key, so I use this one.

```
argocd-repo-server-mtls
  tls.crt
  tls.key
  server-ca.crt
```

Argo CD

Each client's path to server-ca.crt.

```
server.
controller.
applicationsetcontroller.
notificationscontroller.
+ repo.server.ca.cert.path
```



DEMO · THE CLIENTS VERIFY

A client that trusts another CA rejects the repo-server

The action

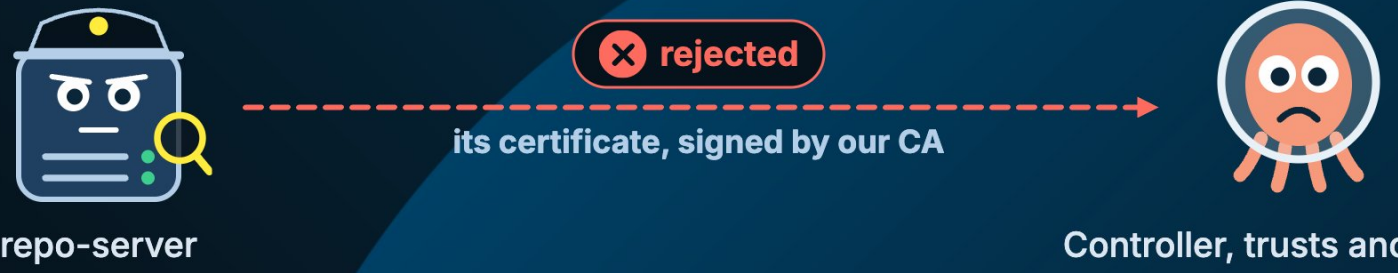
I restart the controller so it trusts a different CA, then put ours back.

The repo-server

No change. It still sends a certificate signed by our CA.

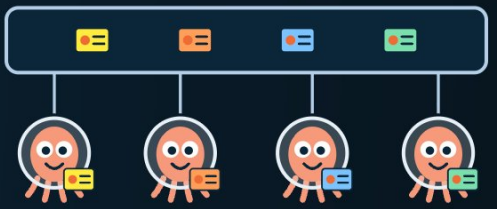
Argo CD

The app reports failed to verify certificate: x509: certificate signed by unknown authority.



OPTION 2 · PER-COMPONENT, ONE SECRET

One Secret for every client's keys



How it works

The Secret stores the CA and a certificate and key for each client. Each client mounts only its own pair, as client.crt and client.key.

What's better

Each component has its own certificate, and all of them are stored in the same secret.

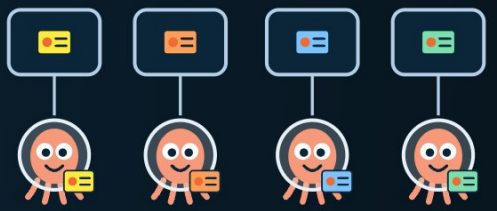
Risks

Anyone who has access to that secret can still read every key.



OPTION 3 · PER-COMPONENT, ONE SECRET EACH

Each client gets its own Secret



How it works

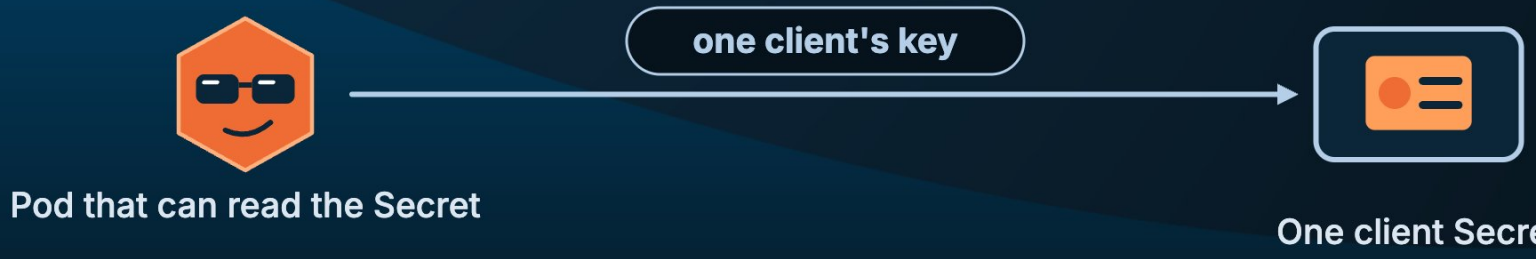
Each client's default volume points at its own Secret with secretName.

What's better

Fine-grained RBAC on each secret, and you can renew each client's certificate on its own schedule.

There's no free beer

You have a Secret per client to create and keep track of, and it's easy to miss one at renewal time.



Source: Argo CD operator manual

DEMO · CLIENT CERTIFICATES

Options 2 and 3 give each client its own certificate

Option 1

One Secret, one certificate

COMMON NAME (CN)

`argocd-core-clients`

on all four clients

Option 2

One Secret, four certificates

COMMON NAME (CN)

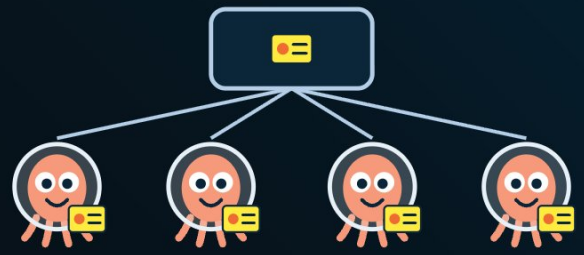
`argocd-server`
`argocd-controller`
`argocd-appset`
`argocd-notifications`

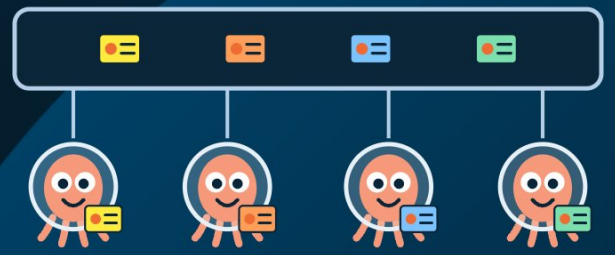
Option 3

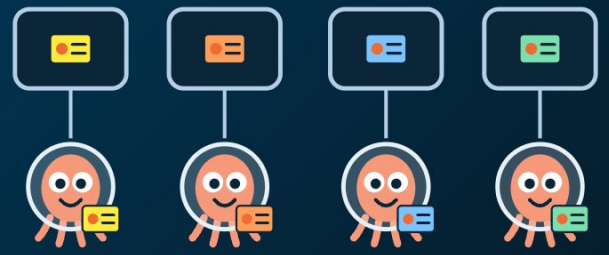
Four Secrets, four certificates

COMMON NAME (CN)

`argocd-server`
`argocd-controller`
`argocd-appset`
`argocd-notifications`







DEMO · A SYNC OVER MTLS

A new commit syncs with all three options

The action

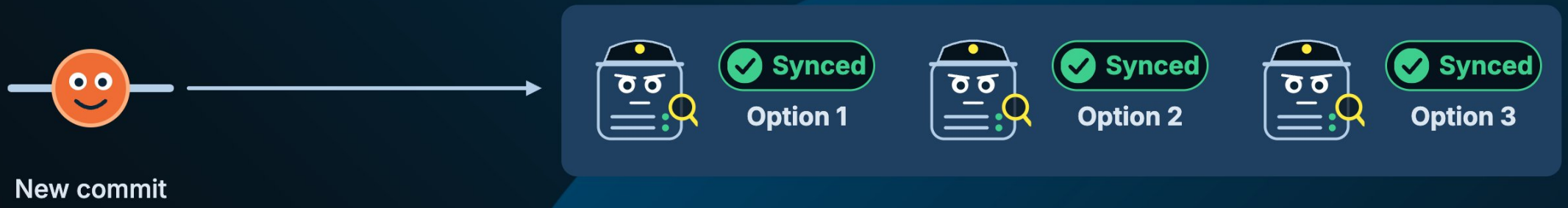
I push a commit to the Git server and ask all three apps to refresh.

The repo-server

It renders the new revision for each controller, over mTLS.

Argo CD

All three apps show Synced on the new revision, and the guestbook ConfigMap shows the new message.



How the three options compare

	1 · Shared cert	2 · One Secret, many keys	3 · Secret per component
Setup effort	Lowest, no patches on the stock manifests	Medium, a volume patch per client	Medium, a one-line patch per client
Identity	One for all clients	One per client	One per client
Rotation	All clients at once	One client at a time, same Secret	One client at a time, own Secret
Secret RBAC	One Secret to protect	One Secret stores all keys	Separate access per Secret
Use it for	Most installs	Single-team installs	Multi-tenant or regulated installs



AUTHORIZATION

The repo-server only checks who signed the certificate



It checks the CA signature and never reads the Subject or SAN.

So anyone with a valid certificate can call any RPC, unless you add your own checks.

ROTATION

Clients load a renewed certificate with no restart, and a new CA needs a repo-server restart

Renewing a client certificate

Update the Secret with a certificate from the same CA. The client reads the new file on its next connection.

Changing the CA

Put both CAs in client-ca.crt and restart the repo-server, give clients certificates from the new CA, then remove the old CA and restart again.

During the restart

The repo-server reads its CA only at startup, so run more than one replica.



DEMO · ROTATION

The renewed certificate is in use with no restart

The action

I renewed the certificate on the last slide, with the same CA and CN. Now I push a commit and follow the new certificate from the Secret to the controller.

The repo-server

It accepts the new certificate, because the CA didn't change.

Argo CD

The controller logs that it reloaded the certificate, no pod restarts, and the app syncs the commit.



FROM TLS TO MTLS

On a running install, update the clients first

Any certificate you already gave the repo-server and any CA paths you set keep working. After these steps the clients still don't check the repo-server unless you set each client's CA path in argocd-cmd-params-cm.

Update the clients

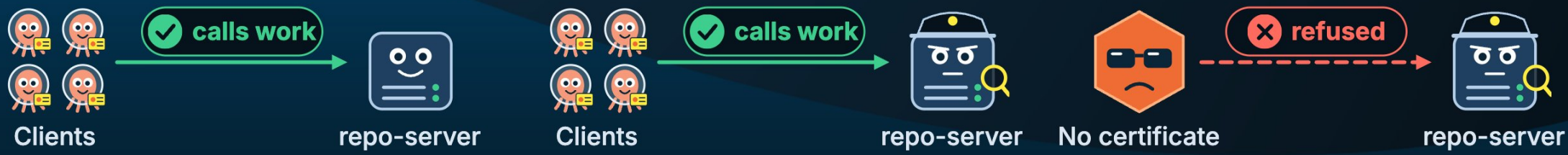
Create argocd-repo-server-mtls with client.crt and client.key only, and restart the four clients.

Update the repo-server

Add client-ca.crt and restart the repo-server. It now refuses any call with no certificate.

Check it

Make a call with no certificate, which should be refused, and sync an app. To go back, remove client-ca.crt and restart the repo-server.



A checklist for your own installation

- Create argocd-repo-server-mtls with the client certificate and restart the clients, then add client-ca.crt and restart the repo-server. The Helm chart's default doesn't mount this Secret, so add that first.
- Start with a shared certificate, and use per-component ones for multi-tenant installs or when each client needs its own key.
- Check for the startup log line mTLS enabled for repo-server, and confirm a call with no certificate is refused.
- Give the repo-server a certificate from your CA and restart it, then set each client's CA path in argocd-cmd-params-cm and restart the clients, so they verify the repo-server.
- mTLS only validates the CA signature, so keep network policy and RBAC.



Links

argo-cd.readthedocs.io/en/latest/operator-manual/mtls · octopus.com/blog/mtls-support-argocd

[Your handle, repo with demo manifests, slides link]