Calico Cloud and argoCD Deploy and Upgrade

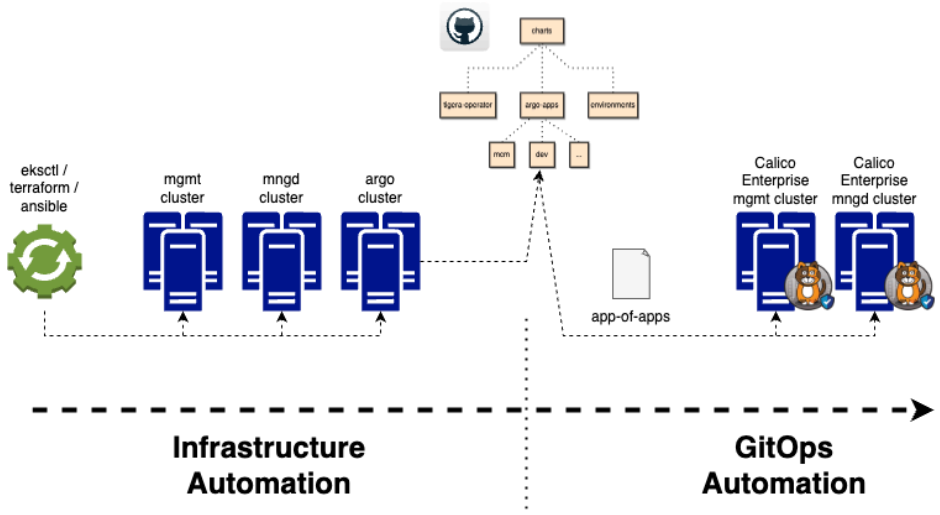
Executive Summary

Most, if not all, Calico Cloud customers are deploying and maintaining their kubernetes clusters via automation. Until recently it was not possible to automate the deployment and upgrade of a Calico Cloud connected cluster but the release of Client Credentials in Calico Cloud fixes this. This document will step through a means to utilize a GitOps approach to automating the lifecycle of a Calico Cloud connected cluster.
At a high-level, the GitOps approach uses a Git repo as a “single source of truth” for the configuration of kubernetes cluster(s). A GitOps tool (such as argoCD or flux) will then read from this single source of truth and then **pull** and apply the stored manifests/configurations to the desired cluster(s). If a change occurs on this single source of truth, GitOps tools will reflect it in an automated or semi-automated fashion onto the production cluster. All aspects of the lifecycle of the solution are managed through this single source of truth allowing teams to collaborate and enabling them to always work on the latest and greatest.

The flow of this document is detailed below

1. (**Theory**) Automating Calico Cloud Deployment
2. (**Practical**) Infrastructure Provisioning
3. (**Theory**) ArgoCD/GitOps Overview
4. (**Practical**) argoCD Provisioning

Automating Calico Cloud Deployment



Automation Overview

The automation of Calico Cloud deployment can be split into two part

i. Infrastructure Automation
ii. GitOps automation

During the Infrastructure Automation workflow we are concerned with deploying the clusters (EKS in this case) and any prerequisites needed to initiate the GitOps automation (e.g. Client Credentials creation, Sealed-Secrets provisioning)
During the GitOps Automation workflow we are concerned with deploying the “applications” that will provision Calico Cloud in our cluster(s) (e.g. Client Credentials provision, Sealed-Secrets operator provisioning, Calico Cloud CRDs and installer provisioning)

Client Credentials

In order to connect a cluster to Calico Cloud, it is necessary to provide an authentication token so that the Calico Cloud management cluster will accept the connection.
Prior to Client Credentials, the only way to generate obtain an authentication token was by requesting one via the Calico Cloud UI
Client Credentials allows a user to create a credential, maybe for a single user or team, and then to associate keys (with variable lifetimes) with this credential. The associated keys can

then be downloaded as kubernetes secrets which can be read by the Calico Cloud Installer and used as the authentication method when connecting to the management cluster.

Infrastructure prerequisites

There are a few prerequisites needed for a successful Calico Cloud automated deployment. Listed below are the prerequisites needed for the deployment

Client Credentials

The client credentials that we will be deploying into the cluster will need to be deployed into a namespace named calico-cloud. You can either pre-create this namespace in the cluster or tell argoCD to create the namespace when it runs the sealed-secret application

Sealed Secrets

It is best practice to encrypt kubernetes secrets when storing them in your git repository. In this guide we will do this by:

1. Following this guide to generate private and public keys that will be used as the “master” secret used by sealed-secrets
2. Create a namespace called sealed-secrets in the *to-be* managed cluster
3. Deploy the “master” secret created in step 1
    a. This will be used by the sealed-secrets controller to decrypted the secrets we will seal and place in the git repository
    b. The sealed-secrets controller will be deployed via an argoCD app later on
4. Seal the client credential and tigera pull secret using the public certificate as per below:
    a. *kubeseal --cert "./${PUBLICKEY}" --scope cluster-wide < mysecret.yaml | kubectl apply -f-*
5. Place the sealed secrets into their respective git repository directories

Infrastructure Provisioning

Cluster provisioning with eksctl

Using the EKS command line tool, *eksctl*, we can provision three clusters that use *kubectl* commands to arrive at the desired state, below we will detail both

High-Level Steps

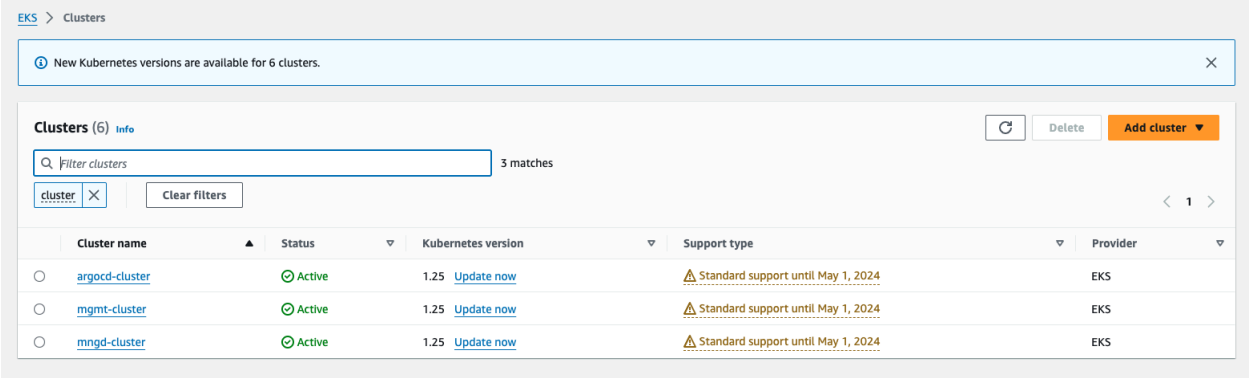
1. Using eksctl, deploy two EKS clusters
    a. A cluster to house our argoCD deployment
    b. A cluster to house the Calico Cloud managed cluster

2. Using kubectl, deploy argoCD and register both clusters, along with the GitHub repository

Outcomes

- EKS infrastructure is setup and argoCD connected and ready

Screenshots



Detailed steps

Management Cluster

1. Run *eksctl create cluster -f mngd-cluster.yaml*

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: mngd-cluster
  region: ca-central-1

nodeGroups:
  - name: leon-node-group # Name as you see fit
    instanceType: t3.large
    desiredCapacity: 3
    volumeSize: 80
    ssh:
      allow: true # will use ~/.ssh/id_rsa.pub as the default ssh key
```

argoCD Cluster

1. Run *eksctl create cluster -f single-cluster.yaml*

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: argocd-cluster
  region: ca-central-1

nodeGroups:
  - name: leon-node-group # Name as you see fit
    instanceType: t3.medium
    desiredCapacity: 1
    volumeSize: 80
    ssh:
      allow: true # will use ~/.ssh/id_rsa.pub as the default ssh key
```

2. Once the cluster has been created, deploy argoCD

```
#!/bin/bash
kubectl config use-context <ARGOCD CLUSTER>
kubectl create ns argocd;
kubectl apply -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install
.yaml -n argocd;
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type":
"LoadBalancer"}}';
sleep 30;
echo -e "-----"
echo  "argo password: " $(kubectl get secret -n argocd
argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d);
echo -e "-----"
echo  "argo service: " $(kubectl get svc -n argocd argocd-server
-ojsonpath='{.status.loadBalancer.ingress[0].hostname}')
sleep 120
argocd login $(kubectl get svc -n argocd argocd-server
-ojsonpath='{.status.loadBalancer.ingress[0].hostname}') --username admin
--password $(kubectl get secret -n argocd argocd-initial-admin-secret -o
jsonpath="{.data.password}" | base64 -d) --insecure
```

3. After the argoCD deployment script has run it will output the argoCD public URL and the password. We will use this information to login to the argoCD cluster via the CLI:

argocd login <url> --username <user> --password <pass>

4. Connect the management and managed clusters to argoCD

```
#!/bin/bash

kubectl config use-context <MANAGED CLUSTER>
CURRENT_CONTEXT=$(kubectl config view --minify -o
jsonpath='{.contexts[0].name}')
CLUSTER_NAME="ce-mngd"
argocd cluster add ${CURRENT_CONTEXT} --name ${CLUSTER_NAME} -y
```

5. Connect the GitHub repository

```
#!/bin/bash
argocd repo add git@github.com:VerodaClient/argocd-charts.git
--ssh-private-key-path=/PATH/TO/PRIVATE/KEY
```

Sealed Secrets Master Secret Provisioning

High-Level Steps

1. Create the “master” sealed-secrets secret
2. Create sealed-secrets namespace
3. Deploy the “master” sealed-secrets secret in the sealed-secrets namespace

Outcomes

- Sealed secrets set up to encrypt secrets outside of the cluster and decrypt inside of the cluster

Detailed steps

Create the “master” sealed-secret secret

General steps can be found here

1. Set up you variables

```
export PRIVATEKEY="mytls.key"
export PUBLICKEY="mytls.crt"
export NAMESPACE="sealed-secrets"
export SECRETNAME="mycustomkeys"
```

2. Generate RSA key pair

```
openssl req -x509 -days 365 -nodes -newkey rsa:4096 -keyout "$PRIVATEKEY"
-out "$PUBLICKEY" -subj "/CN=sealed-secret/O=sealed-secret"
```

3. Create the sealed-secret “master” secret and deploy into the managed cluster

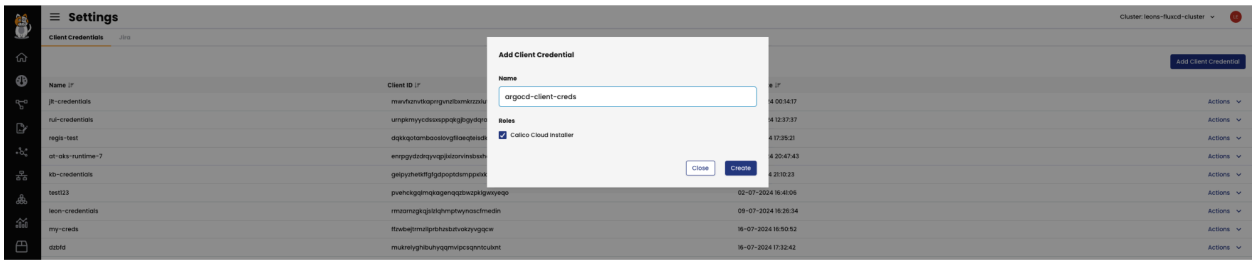
```
kubectl config use-context <mng-cluster-context>
kubectl create ns "$NAMESPACE"
kubectl -n "$NAMESPACE" create secret tls "$SECRETNAME" --cert="$PUBLICKEY"
--key="$PRIVATEKEY"
kubectl -n "$NAMESPACE" label secret "$SECRETNAME"
sealedsecrets.bitnami.com/sealed-secrets-key=active
```

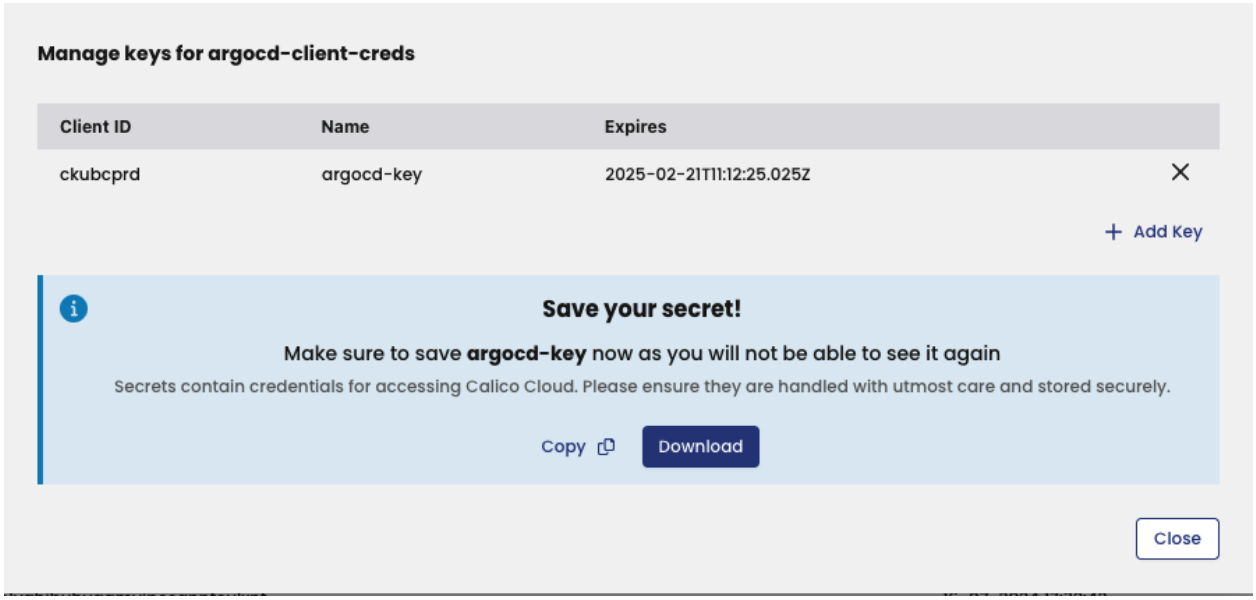
Client Credentials Creation

High-Level Steps

1. Generate a Client Credential and download the associated key
2. Seal the Client Credential secret

Screenshots





Outcomes

- Client Credentials encrypted, ready to be deployed into the cluster

Detailed steps

Generate the Client Credentials

1. Login to the Calico Cloud UI with an Admin user
2. From the user button on the top-right hand side select **Settings**
3. In the Client Credentials page click “**Add Client Credentials**”
4. Give the Client Credentials a name and ensure that the “Calico Cloud Installer” role is selected
5. On the newly created Client Credentials, select **Actions** -> **Manage Keys**
6. Select **+ Add Key**
7. Enter a name for the key and an expiration date
8. Download the secret

Seal the Client Credentials

1. Locate the Client Credential secret yaml that was downloaded in the previous step
2. Run the below command to seal this secret

```
kubeseal --cert "./${PUBLICKEY}" --scope cluster-wide <
client-credential-secret.yaml > sealed-client-credential-secret.yaml
```

3. This sealed secret is now ready to be added to our Git repo into the **charts/client-credentials/dev-team-1-creds/templates** directory

ArgoCD/GitOps Overview

GitOps Principles

GitOps is a set of practices that uses Git as the single source of truth for managing infrastructure and application configurations. In a GitOps workflow, all changes are made **declaratively** in Git repositories, and those changes are automatically **pulled** and synchronized to the target environment using tools like Argo CD. Argo CD is a continuous delivery tool specifically designed for Kubernetes, which monitors Git repositories for changes and ensures that the deployed state of a cluster matches the desired state defined in Git. This approach promotes consistency, auditability, and reliability by enabling automated deployments and rollbacks, all managed through Git.

ArgoCD Strategies Used

App-of-Apps Pattern

An ArgoCD App-of-Apps pattern is a GitOps approach for managing Kubernetes applications. It involves structuring the entire application infrastructure as a Git repository, with a central "app-of-apps" manifest orchestrating the deployment of multiple applications. This top-level “parent” manifest streamlines the process of updating the Kubernetes cluster automatically in response to changes in the Git repository. This pattern simplifies the management of complex multi-application environments, promotes version control, and facilitates collaboration between development and operations teams by codifying the entire application stack in a versioned and auditable manner.

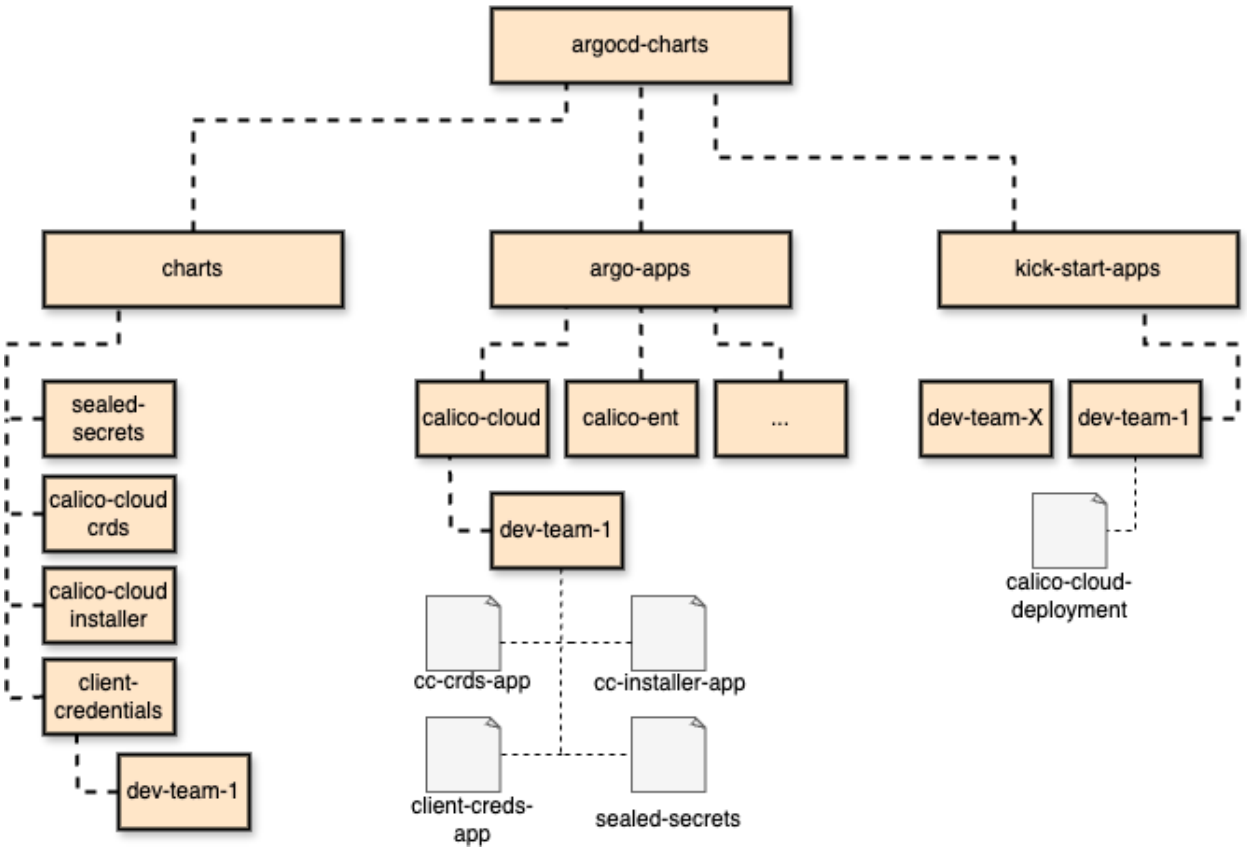
Syncwaves

Syncwaves is a method of ordering what order applications are run in. Simply put, the lower the syncwave, the earlier an application will run and higher numbered syncwave applications will need to wait.

Cluster Architecture

Cluster Name	Cluster Type	Cluster Function	Number of nodes	argoCD App
argo-cd-cluster	EKS	argoCD deployment	1	app-of-apps
mgmt-cluster	EKS	CC managed cluster	3	calico-cloud-deployment.yaml

Github Repo Architecture



Repo Architecture

Git repository

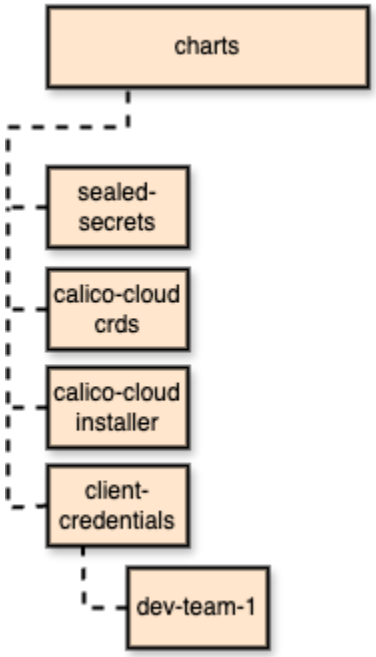
https://github.com/VerodaClient/argocd-charts.git
(The repo is private so please request access)

Overview

This repo architecture is designed in a way that tries to split different functions under specific directories
charts - This contains all helm charts that can be consumed but argoCD applications.
Think of these as the “ingredients”
argo-apps - This contains all of the argoCD applications that individually point to the helm charts in the charts directory, and together represent a full deployment (e.g. deploying Calico Cloud to a cluster).
Think of these as the “recipe”
kick-start-apps - This contains the app-of-apps argoCD application that calls the argoCD applications housed in the argo-apps directory. Its job is to monitor and make sure that all argoCD applications that together represent a full deployment are all in sync.

Think of this as the “chef”

Charts



The charts directory contains all of the helm charts that will be used by argoCD apps when deploying into clusters. These charts are meant to be shared and reused across argoCD applications and teams within the organization.

Sealed Secrets

This contains the helm chart for Bitnamis sealed-secrets. This will be used to deploy the sealed-secrets controller that will in turn be used to unseal the sealed-client-credentials. Nothing has been customized

Calico-cloud-crds

This contains the helm chart for all of the Calico Cloud CRDs (calico-cloud-crds) needed to connect a managed cluster to calico-cloud. Pulled via https://installer.calicocloud.io/charts.

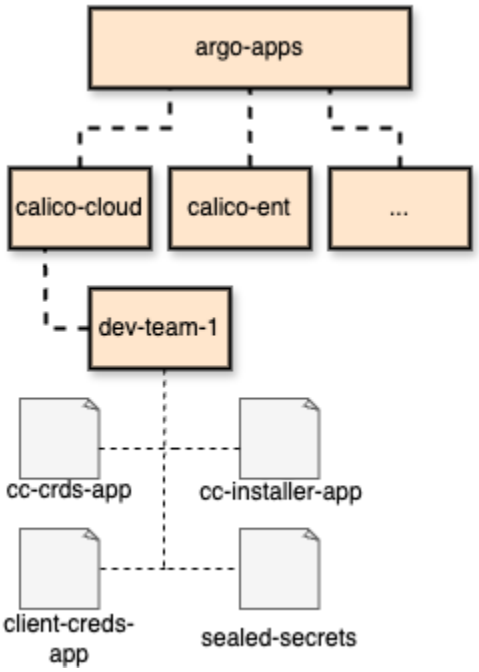
Calico-cloud-installer

This contains the helm chart for the Calico Cloud Installer (calico-cloud) needed to connect a managed cluster to calico-cloud. Pulled via https://installer.calicocloud.io/charts.

Client-credentials

This contains the secret credentials needed to automate the deployment of Calico Cloud

argo-apps



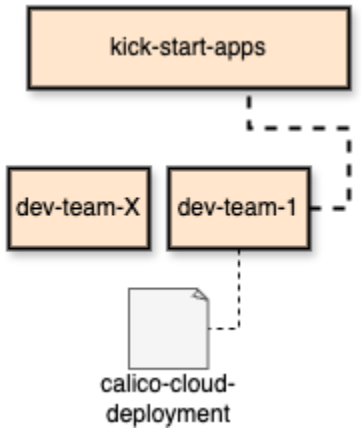
The argo-apps directory contains all of the argoCD applications that will be used by argoCD to deploy a solution (e.g. deploy calico-cloud, calico-enterprise, business applications etc). These argoCD applications will consume the helm charts within the charts folder to build these solutions.

calico-cloud/dev-team-1

This solution will deploy calico cloud onto a cluster by installing four helm charts

- sealed-secrets - Used to unseal/decrypt the client-credentials
- client-creds - Used by the calico cloud installer to authorize pulling the calico-cloud images and deploying to the cluster
- cc-crds - Deploys the relevant Calico Cloud CRDs
- cc-installer - Deploys the Calico Cloud installer deployment that kicks off the calico-cloud deployment

kick-start-apps



The kick-start-apps directory contains all of the argoCD applications that will call the argo-apps directories. The kick-start-apps are known as **app-of-app**s in that they are a parent argoCD application that deploys, syncs and monitors the child apps in the specified argo-apps directory. If a child argoCD application is deleted or changed, outside of argoCD, it is this parent app that will re-synchronize to ensure a consistent state.

Syncwaves

app-of-apps (No syncwave assigned)

This is the “parent” application that will call all of the other argo applications located in argo-apps/calico-cloud/dev-team-x directory.

sealed-secret (syncwave 1)

This application points to the ***charts/sealed-secrets*** directory. This needs to be deployed first so that all of our sealed secrets (client credentials) can be unsealed/decrypted, ready for use

client-credentials (syncwave 2)

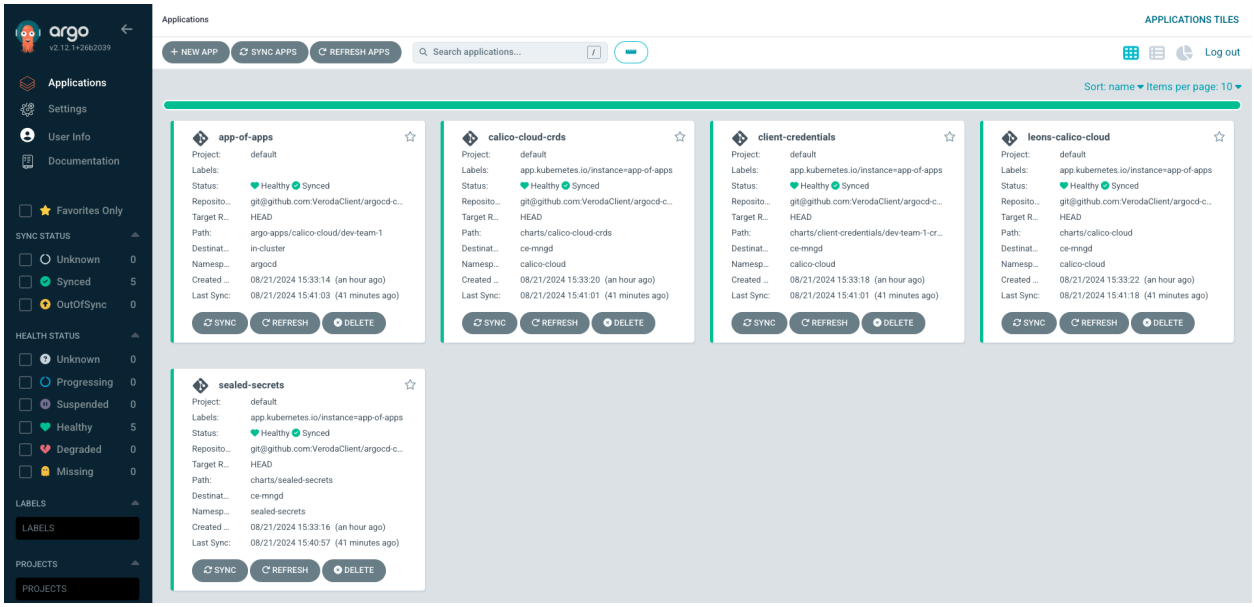
This application points to the ***charts/client credentials*** directory. This directory contains the sealed/encrypted client-credentials. Once deployed it will be unsealed by the sealed-secrets controller so that it can be used to pull images by the calico-cloud installer

calico-cloud-crds (syncwave 3)

This application points to the ***charts/calico-cloud-crds*** directory. This directory contains the helm chart used to install the CRDs needed for calico-cloud.

calico-cloud-installer (syncwave 4)

This application points to the ***charts/calico-cloud*** directory. This directory contains the calico-cloud installer helm chart.



argoCD Provisioning

App-of-Apps Pattern

In this guide we will create an app-of-apps application (detailed below) that will then call all of the argoCD applications needed to deploy Calico Cloud.
The only application that needs to be added to the argoCD deployment is **kickstart-apps/dev-team-1/calico-cloud-deploy.yaml**. After this, everything else will be taken care of

app-of-apps.yaml

Application

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app-of-apps
  annotations:
    argocd.argoproj.io/sync-wave: "-100"
```

```yaml
spec:
  destination:
    name: in-cluster
    namespace: argocd
  source:
    path: argo-apps/calico-cloud/dev-team-1
    repoURL: 'git@github.com:VerodaClient/argocd-charts.git'
    targetRevision: HEAD
  project: default
  syncPolicy:
    automated:
      selfHeal: true
  syncWaves:
  - name: wave1
    applications:
    - sealed-secrets
  - name: wave2
    applications:
    - client-credentials
  - name: wave3
    applications:
    - calico-cloud-crds
  - name: wave4
    applications:
    - calico-cloud
```

High-Level Steps (UI)

1. In argoCD select “Create New Application”
2. Select “Edit as Yaml”
3. Copy and Paste the above application
4. Click “Save”
5. Click “Create”

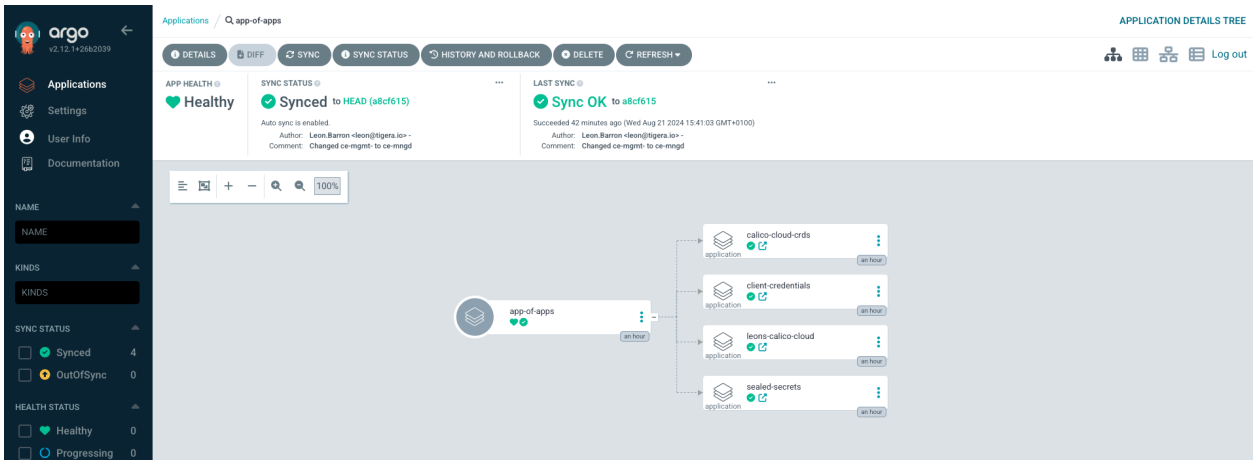
High-Level Steps (CLI)

1. argocd app create -f **kickstart-apps/dev-team-1/calico-cloud-deploy.yaml**

Outcomes

1. The app-of-app application will deploy all other necessary applications

Screenshots



Details

This application is the “parent” application that will deploy all other needed applications to setup a Calico Enterprise MCM cluster. It is the only application that is needed to set everything up, assuming the git repository structure is as described above

sealed-secret

Application

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: sealed-secrets
  annotations:
    argocd.argoproj.io/sync-wave: "-10"
spec:
  project: default
  destination:
    name: ce-mngd
    namespace: sealed-secrets
  source:
    path: charts/sealed-secrets
```

```
    repoURL: 'git@github.com:VerodaClient/argocd-charts.git'
    targetRevision: HEAD
  syncPolicy:
    automated:
      selfHeal: true
    retry:
      limit: 35
      backoff:
        duration: 30s
        factor: 1
        maxDuration: 3m
    syncOptions:
      - Validate=true
      - CreateNamespace=true
      - ApplyOutOfSyncOnly=true
      - ServerSideApply=true
      - RespectIgnoreDifferences=true
  ignoreDifferences:
    - group: apiextensions.k8s.io
      kind: CustomResourceDefinition
      name: beats.beat.k8s.elastic.co
      jsonPointers:
        - /spec
    - kind: PersistentVolume
      jsonPointers:
        - /spec/claimRef/resourceVersion
        - /spec/claimRef/uid
        - /status/lastPhaseTransitionTime
```

Outcomes

1. Sealed Secret Controller is deployed into the managed cluster
2. Sealed Secret Controller will use the “master” sealed secret that we previously deployed into the sealed-secrets namespace

client-credentials

Application

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: client-credentials
  annotations:
    argocd.argoproj.io/sync-wave: "0"
spec:
  project: default
  destination:
    name: ce-mngd
    namespace: calico-cloud
  source:
    path: charts/client-credentials/dev-team-1-creds
    repoURL: 'git@github.com:VerodaClient/argocd-charts.git'
    targetRevision: HEAD
  syncPolicy:
    automated:
      selfHeal: true
    retry:
      limit: 35
      backoff:
        duration: 30s
        factor: 1
        maxDuration: 3m
    syncOptions:
      - Validate=true
      - CreateNamespace=true
      - ApplyOutOfSyncOnly=true
      - ServerSideApply=true
      - RespectIgnoreDifferences=true
  ignoreDifferences:
    - group: apiextensions.k8s.io
      kind: CustomResourceDefinition
      name: beats.beat.k8s.elastic.co
      jsonPointers:
        - /spec
    - kind: PersistentVolume
      jsonPointers:
        - /spec/claimRef/resourceVersion
```

```
        - /spec/claimRef/uid
        - /status/lastPhaseTransitionTime
```

Outcomes

1. A sealed/encrypted client-credentials secret is deployed into the tigera-operator namespace
2. This will be unsealed/decrypted by the sealed secrets controller and will then be used by the calico cloud installer to pull the calico cloud images

cc-crds

Application

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: calico-cloud-crds
  namespace: argocd
  annotations:
      argocd.argoproj.io/sync-wave: "1"
spec:
  project: default
  source:
    path: charts/calico-cloud-crds
    repoURL: 'git@github.com:VerodaClient/argocd-charts.git'
    targetRevision: HEAD
    helm:
      valueFiles:
      - values.yaml
  destination:
    name: ce-mngd
    namespace: calico-cloud
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Outcomes

1. Calico cloud CRDs are deployed into the cluster based on the helm chart located in charts/calico-cloud-crds

cc-installer

Application

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: leons-calico-cloud
  namespace: argocd
  annotations:
      argocd.argoproj.io/sync-wave: "2"
spec:
  project: default
  source:
    repoURL: 'git@github.com:VerodaClient/argocd-charts.git'
    path: charts/calico-cloud
    targetRevision: HEAD
    helm:
      releaseName: "calico-cloud"
      valueFiles:
      - values.yaml
  destination:
    name: ce-mngd
    namespace: calico-cloud
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Outcomes

1. Calico-cloud installer deployed into the managed cluster and the Calico Cloud deployment is kicked off
2. The values.yaml will also customize this deployment

- Set the cluster name to be "leons-argocd-cluster" (Please use a name that is not already in use in connected clusters in calicocloud.io)
- Sets the calico-cloud version to 19.3.0
- Disables certain features:
  - Image Assurance
  - Security Posture (Dashboard)
  - Runtime Security (Container Threat Detection)

Screenshots (values.yaml)

```
installer:
  # ClusterName is the name this cluster should appear as in the Calico
Cloud UI.
  # The value passed here should match the name used when creating the
Managed Cluster in
  # the Calico Cloud UI.
  # Note that ClusterName cannot be modified once it is set.
  # Required field.
  clusterName: "leons-argocd-cluster"

  calicoCloudVersion: "v19.3.0"
  # ResourceVersion is a random value that, when modified, will force a
reinstall.
  resourceVersion: ""

  # UploadDiags enables automatic upload of diagnostics bundles to Calico
Cloud, viewable by the support team.
  uploadDiags: false

  # registry specifies a private registry to use for all images.
  registry: ""

  # imagePath is an optional field which can be used to control the
'subdirectories'
  # where Tigera images are located in the private registry.
  imagePath: ""

  components:
    imageAssurance:
      state: Disabled
    securityPosture:
      state: Disabled
    runtimeSecurity:
```

```
      state: Disabled

# apiKey is the key generated in the Calico Cloud UI when adding a new
managed cluster.
# Required field for installs initiated from the Calico Cloud UI.
apiKey: ""

image:
  # Overrides the image tag
  tag: ""

  pullPolicy: IfNotPresent

imagePullSecrets: []

resources:
  limits:
    cpu: 250m
    memory: 300Mi
  requests:
    cpu: 200m
    memory: 200Mi

# caBundleSecretName is the name of the secret in the calico-cloud
namespace to use as CA certificates.
# it should contain key named "ca-certificates.crt".
# it will be mounted into the pod at /etc/ssl/certs
caBundleSecretName: ""
```

Calico Cloud Upgrade

Upgrading Calico Cloud Versions using argoCD

High-Level Steps

1. Change the value **calicoCloudVersion** in the charts/calico-cloud/values.yaml file to the Calico Cloud version you desire (must be an upgrade, downgrade is not supported)
2. Wait for argoCD to pick up this change or, alternatively, refresh all argoCD apps

Outcomes

1. Calico cloud is upgraded to the version specified in the charts/calico-cloud/values.yaml file

Detailed Steps

1. Navigate to charts/calico-cloud/values.yaml
2. Edit this file to change the value **calicoCloudVersion** to the required Calico Cloud version
3. Refresh (may need a hard refresh) the argoCD apps in argoCD UI

Future Improvements

1. TBD

Troubleshooting

- If you see the apps in argoCD coloured in grey ensure that:
    - The git repo is successfully connected
    - The cluster is successfully connected
    - The argoCD applications are targeting the correct cluster
        - E.g., they should be targeting a cluster named ce-mngd and **not** ce-mgmt

Gaps/RFEs

- TBD