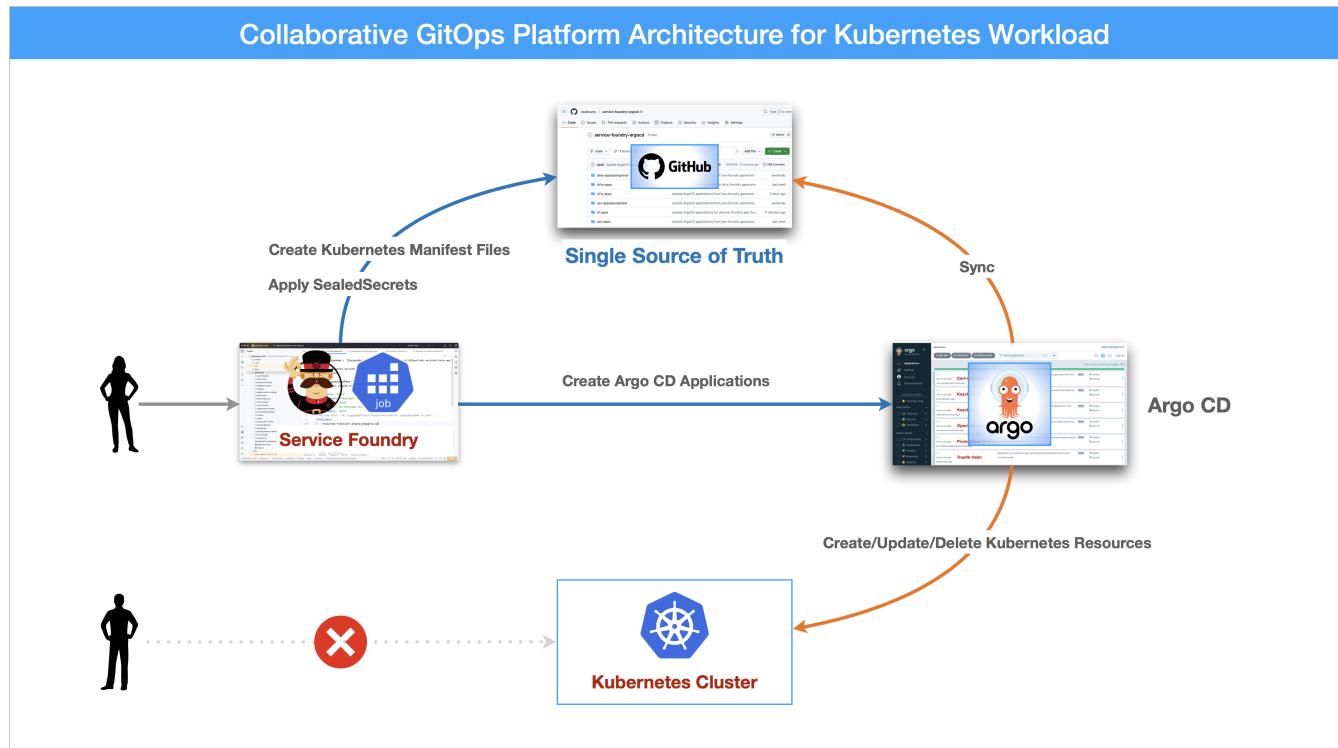


Service Foundry: User Guide

Young Gyu Kim <credemol@gmail.com>



Helping you manage your Kubernetes apps with ease - no deep tech knowledge needed!

End-to-End GitOps Orchestration for Kubernetes Workloads

A Modular GitOps Framework for Kubernetes-Native Platform Engineering

What Is Service Foundry?

Service Foundry is a Kubernetes-native platform engineering framework designed to help teams provision, deploy, secure, and observe workloads using GitOps workflows.

It integrates popular open-source tools into a cohesive automation system, enabling you to:

- Use standardized GitOps patterns to manage infrastructure and workloads
- Collaborate through version-controlled configuration
- Deploy complete application stacks (e.g., observability, identity, data pipelines)
- Secure resources using Single Sign-On (SSO) and encrypted secrets

It bridges the gap between developers and operations by providing a **UI-driven GitOps console**, while preserving the **auditable, declarative** nature of Git-based infrastructure as code.

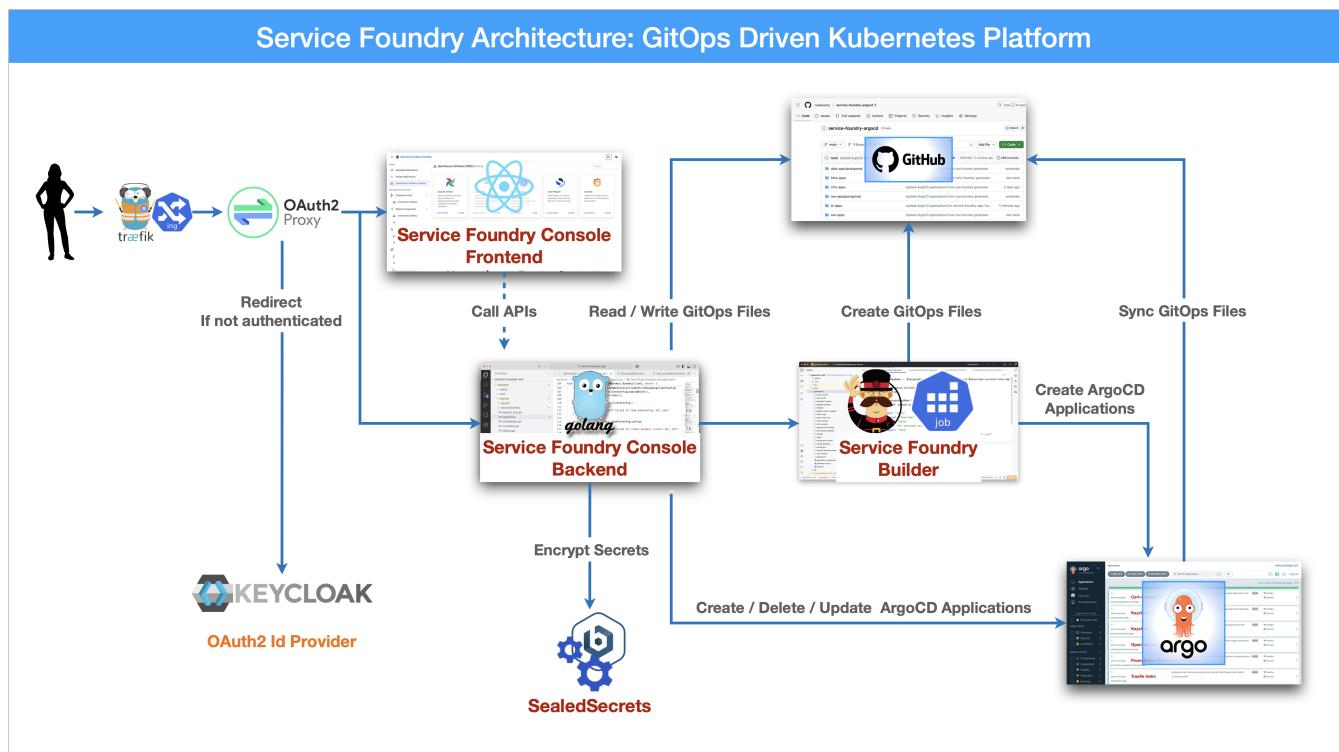


Figure 1. GitDps-Driven Kubernetes Platform

GitOps-Driven Platform Management

Service Foundry adopts a GitOps-first approach, meaning:

- All Kubernetes resources (infra modules, apps, secrets, etc.) are declared in YAML and stored in Git
- Changes are reviewed via Pull Requests and automatically applied by Argo CD
- The entire state of the cluster is version-controlled, auditable, and reproducible

This enables:

- Predictable deployments — Git is the single source of truth
- Safe rollbacks — Easily revert to previous versions
- Transparent ops — Operational changes are traceable via Git history

Core Components of Service Foundry

Table 1. Service Foundry Components

Component	Purpose
Argo CD	Reconciles Kubernetes state with Git
SealedSecrets	Encrypts secrets for safe Git storage
Keycloak	Acts as the OAuth2 identity provider (SSO)
Traefik	Kubernetes-native ingress and routing
Oauth2 Proxy	Authentication middleware for UIs
Service Foundry Console	Web UI to generate, review, and manage GitOps manifests
Service Foundry Builder	CLI/Backend tool to generate manifests and bootstrap environments
Source Generator	Generates Kubernetes manifests from templates

Why Use Service Foundry?

- **Declarative + Automated:** All infrastructure and app configurations are versioned and applied automatically
- **Built-in Security:** Seamless OAuth2-based login, secret encryption, and role-based access
- **Production-Grade Observability:** OpenTelemetry, Prometheus, Grafana, and Loki are pre-integrated
- **Unified Management Interface:** Web console for all configuration and lifecycle tasks
- **Modular Architecture:** Deploy only the modules you need — SSO, Observability, Big Data, etc.
- **Developer & Platform Collaboration:** Developers install services; operators manage infrastructure — all within GitOps

Rapid Bootstrap with a Single Command

Service Foundry provides a CLI tool and Helm charts to **bootstrap your platform in one step**:

- Installs Argo CD, Keycloak, Traefik, SealedSecrets, and Service Foundry Console
- Sets up initial Git repository structure
- Deploys infrastructure modules and app scaffolds using GitOps
- Secures access with SSO

Installation Command using Helm

```
$ helm install service-foundry-builder \
  service-foundry/service-foundry-builder \
  --set command=bootstrap \
  -n service-foundry --create-namespace \
  --version $SF_BUILDER_CHART_VERSION
```

After setup, platform users can log in to the console and provision applications from a form-based UI — with YAML manifests automatically committed to Git.

Service Foundry Console

The Service Foundry Console provides a self-service web interface for platform users to provision and manage workloads. By default, it is accessible at:

<http://sfapp.{your-root-domain}>

NOTE Ensure that your DNS is configured to point to the Traefik load balancer.

The screenshot shows the Service Foundry Console interface. On the left is a sidebar with a tree view of categories: Git Ops (Managed Applications, Enterprise Applications, OpenSource Software, GitOps Applications), K8s Application Orchestrator (Framework Core, Shared Components, Observability, Single Sign On (SSO)), Spring Backend, Big Data, and Settings (Settings). The main area is titled "Managed Applications" and lists ten applications. Each application row includes a checkbox, the application name (prefixed with "argocd / sf-"), namespace, creation timestamp, version, health status (green heart icon), sync status (blue circular icon with a checkmark), and an "Action" button. The applications listed are: argocd / sf-keycloak-24.4.13-helm-app, argocd / sf-keycloak-24.4.13-kustomize-app, argocd / sf-oauth2-proxy-helm-app, argocd / sf-oauth2-proxy-kustomize-app, argocd / sf-sealed-secrets-0.30.0-kustomize-app, argocd / sf-service-foundry-app-backend-helm-app, argocd / sf-service-foundry-app-frontend-helm-app, argocd / sf-service-foundry-app-frontend-kustomize-app, argocd / sf-traefik-34.4.1-helm-app, and argocd / sf-traefik-ingress-kustomize-app. The top right corner shows "Deployment Job Status - Start: 9:03:10 PM Completion: 9:09:36 PM Status: Succeeded".

Figure 2. Service Foundry Console

All components, including the console itself, are deployed as **Argo CD Applications**. Their manifests are stored and versioned in the GitOps repository, ensuring full traceability and reproducibility. Users can operate either via the console or directly using Git workflows.

Git Repository

The Git repository initialized during bootstrap stores all Kubernetes manifest files managed by

Service Foundry — including infrastructure modules, Argo CD Applications, Helm values files, SealedSecrets and Kubernetes Resources.

The screenshot shows a GitHub repository interface for 'service-foundry-argocd'. The repository has a main branch named 'main'. A commit titled 'nsa2 Update ArgoCD applications for service-foundry-app-frontend and servi...' was made 8 minutes ago. The commit message indicates updates from various generators: argocd, keycloak, oauth2-proxy, sealed-secrets, service-foundry-app-backend, service-foundry-app-frontend, traefik-ingress, and traefik. The commit details show the last commit message and date for each component.

Name	Last commit message	Last commit date
...		
argocd/sf-argocd-8.1.2-kustomize-app/post	Update ArgoCD applications from argo-foundry generator	13 minutes ago
keycloak	Update ArgoCD applications from infra-foundry generator	11 minutes ago
oauth2-proxy	Update ArgoCD applications from sso-foundry generator	8 minutes ago
sealed-secrets	Update ArgoCD applications from argo-foundry generator	13 minutes ago
service-foundry-app-backend	Update ArgoCD applications for service-foundry-app-frontend and servi...	8 minutes ago
service-foundry-app-frontend	Update ArgoCD applications for service-foundry-app-frontend and servi...	8 minutes ago
traefik-ingress	Update ArgoCD applications from sso-foundry generator	8 minutes ago
traefik	Update ArgoCD applications from infra-foundry generator	11 minutes ago

Figure 3. GitOps Repository with Initial Structure

This centralized structure supports both manual Git operations (clone, commit, push, etc.) and visual editing via the Service Foundry Console.

SealedSecrets for Secure Configuration

Sensitive data (e.g., passwords, tokens, API keys) are encrypted using Bitnami SealedSecrets before being committed to Git. During bootstrap, Service Foundry automatically converts Kubernetes Secret manifests into SealedSecret manifest.

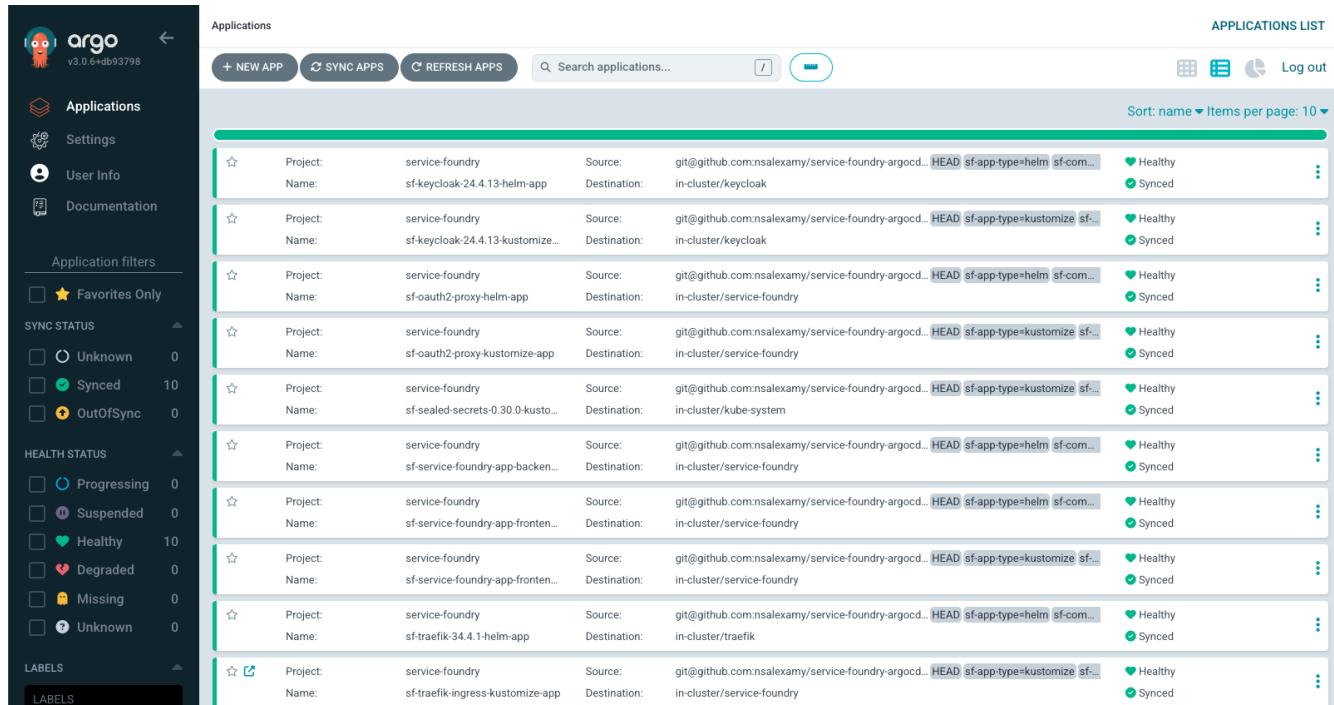
The screenshot shows a GitHub repository interface for 'service-foundry-argocd'. The repository has a main branch named 'main'. A commit titled 'nsa2 Update ArgoCD applications from infra-foundry generator' was made 42 minutes ago. The commit message indicates an update from the infra-foundry generator. The commit details show the last commit message and date. Below the commit, a file named 'keycloak-credentials-secret-24.4.13.yaml' is displayed. The file content is a YAML representation of a SealedSecret manifest, showing fields like kind, apiVersion, metadata, spec, template, and encryptedData.

```
1  {
2    "kind": "SealedSecret",
3    "apiVersion": "bitnami.com/v1alpha1",
4    "metadata": {
5      "name": "keycloak-credentials",
6      "namespace": "keycloak",
7      "creationTimestamp": null
8    },
9    "spec": {
10      "template": {
11        "metadata": {
12          "name": "keycloak-credentials",
13          "namespace": "keycloak",
14          "creationTimestamp": null
15        }
16      },
17      "encryptedData": {
18        "admin-password": "AgAwMELe0gws0DqmUqpeQJZWxQ5R/h0yrQ4AuOCa3wa18C2wi3bcg+Acbk4dQezU3mxwEpwGDjmtNwQiHChiv61RyEQPO0SlCo0P6QsV/QbcXacNnxhJz1rA5BDTdt/ECSTOR+Sq/tMzu22Qd2PK+7i+SM8PuT/C"
19      }
20    }
21 }
```

Figure 4. SealedSecrets Applied

Argo CD Integration

All components of the Service Foundry platform, Open Source Software applications, and user-defined applications are managed as Argo CD Applications. This ensures that the desired state defined in Git is continuously reconciled with the actual state in the Kubernetes cluster.



The screenshot shows the Argo CD Applications interface. On the left is a sidebar with navigation links: Applications, Settings, User Info, Documentation, Application filters (Favorites Only, SYNC STATUS, HEALTH STATUS, LABELS), and a Labels section. The main area is titled 'Applications' and contains a table with 12 rows. Each row represents an application with columns for Project, Name, Source, Destination, Status (Healthy/Synced), and a more options menu. The table has a header with 'Sort: name ▾ Items per page: 10 ▾' and icons for grid, list, and search.

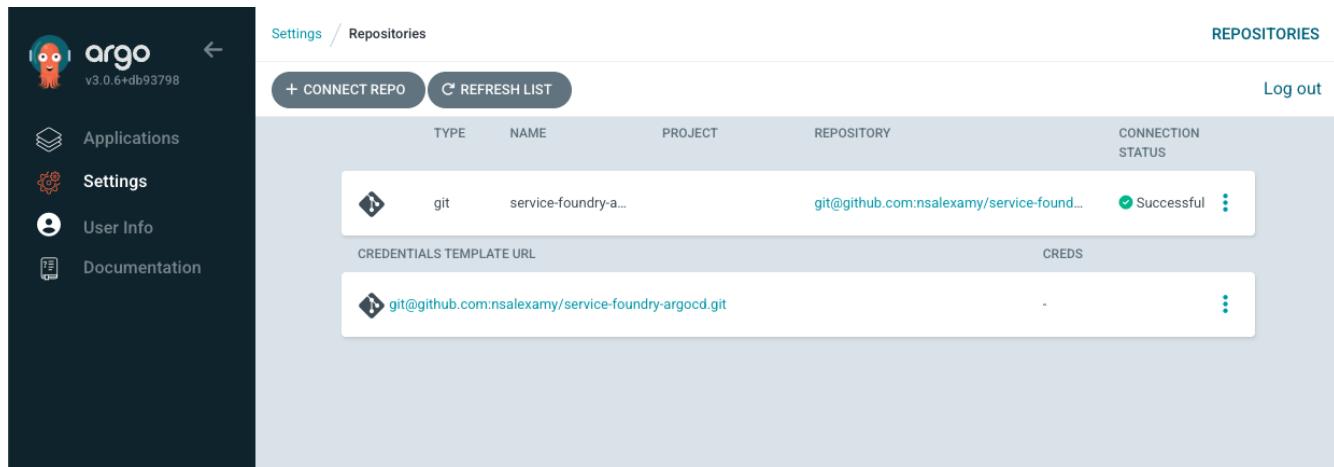
Project	Name	Source	Destination	Status	More
service-foundry	sf-keycloak-24.4.13-helm-app	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=helm sf-com...	in-cluster/keycloak	Healthy Synced	⋮
service-foundry	sf-keycloak-24.4.13-kustomize-app	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=kustomize sf-com...	in-cluster/keycloak	Healthy Synced	⋮
service-foundry	sf-oauth2-proxy-helm-app	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=helm sf-com...	in-cluster/service-foundry	Healthy Synced	⋮
service-foundry	sf-oauth2-proxy-kustomize-app	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=kustomize sf-com...	in-cluster/service-foundry	Healthy Synced	⋮
service-foundry	sf-sealed-secrets-0.30.0-kusto...	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=kustomize sf-com...	in-cluster/kube-system	Healthy Synced	⋮
service-foundry	sf-service-foundry-app-backend	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=helm sf-com...	in-cluster/service-foundry	Healthy Synced	⋮
service-foundry	sf-service-foundry-app-fronten...	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=helm sf-com...	in-cluster/service-foundry	Healthy Synced	⋮
service-foundry	sf-service-foundry-app-fronten...	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=kustomize sf-com...	in-cluster/service-foundry	Healthy Synced	⋮
service-foundry	sf-traefik-34.4.1-helm-app	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=helm sf-com...	in-cluster/traefik	Healthy Synced	⋮
service-foundry	sf-traefik-ingress-kustomize-app	git@github.com:nsalexamy/service-foundry-argocd.. HEAD sf-app-type=kustomize sf-com...	in-cluster/service-foundry	Healthy Synced	⋮

Figure 5. Argo CD Applications

The GitOps Repository is configured in Argo CD Settings as a repository, allowing Argo CD to sync the applications defined in the repository to the Kubernetes cluster.

Repository Configuration

Service Foundry automatically registers your GitOps repository in Argo CD during bootstrap, allowing continuous sync of application states.



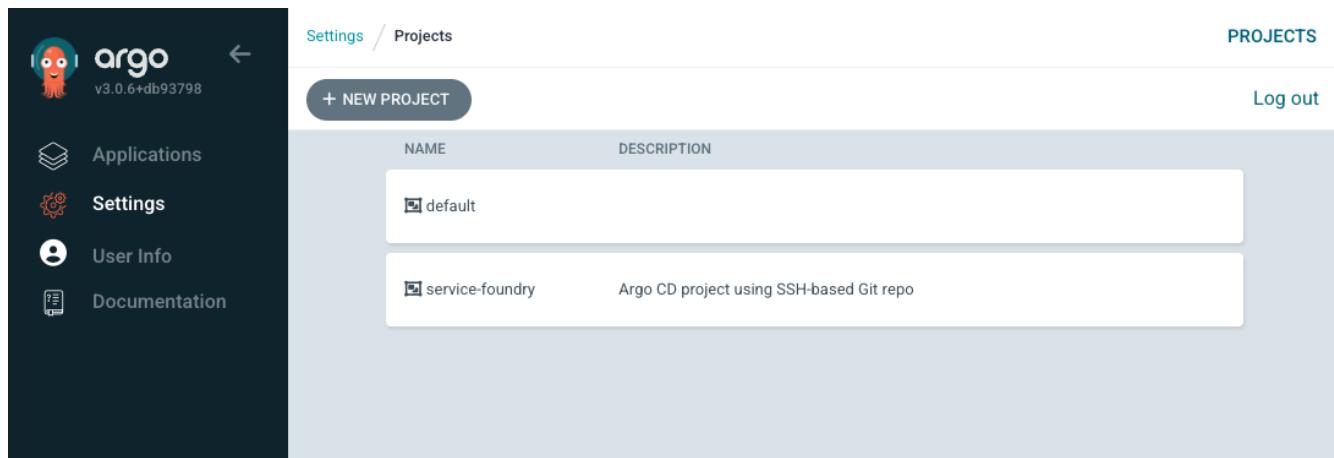
The screenshot shows the Argo CD Repositories interface. On the left is a sidebar with navigation links: Applications, Settings, User Info, Documentation. The main area is titled 'Settings / Repositories' and contains a table with one row. The table has columns for TYPE, NAME, PROJECT, REPOSITORY, and CONNECTION STATUS. Below the table is a section for CREDENTIALS TEMPLATE URL with a field containing 'git@github.com:nsalexamy/service-foundry-argocd.git'. There is also a 'Log out' link at the top right.

TYPE	NAME	PROJECT	REPOSITORY	CONNECTION STATUS
git	service-foundry-a...		git@github.com:nsalexamy/service-foundry-argocd.git	Successful ⋮

Figure 6. Argo CD Repositories

Argo CD Project Scope

A dedicated Argo CD Project named service-foundry is created by default to scope all applications deployed via Service Foundry.

A screenshot of the Argo CD web interface. The top navigation bar includes 'Settings' and 'Projects'. A sidebar on the left has icons for 'Applications', 'Settings', 'User Info', and 'Documentation'. The main area shows a table with two rows. The first row is for 'default' with no description. The second row is for 'service-foundry' with the description 'Argo CD project using SSH-based Git repo'. A 'Log out' button is in the top right corner.

NAME	DESCRIPTION
default	
service-foundry	Argo CD project using SSH-based Git repo

Figure 7. Argo CD Projects

Service Foundry Console Features

The **Service Foundry Console** provides a visual control plane for managing, observing, and operating applications and platform components deployed across your Kubernetes clusters. It offers a simplified interface for interacting with GitOps-managed resources and platform services.

GitOps-Centric Application Management

- **Managed Applications:** View the full list of Argo CD-managed applications, monitor their sync and health status, and perform lifecycle operations such as uninstall or update.
- **Enterprise Applications:** Deploy and manage proprietary or internal software packages, typically sourced from private Helm registries.
- **Open Source Software:** Browse and install curated open-source packages (e.g., Redis, Postgres, Kafka) with Helm chart-based GitOps workflows.
- **Raw GitOps Applications:** Directly edit or remove raw Argo CD Applications stored in the GitOps repository.

Kubernetes Stack Orchestration

- **Framework Core:** Inspect and manage foundational components installed during platform bootstrap (e.g., Argo CD, Traefik, SealedSecrets).
- **Shared Components:** Control cluster-wide operators and services that support multiple stacks — such as Prometheus Operator, OpenTelemetry Operator, Spark Operator.
- **Observability Stack:** Manage observability tools like Grafana, Prometheus, Loki, Tempo, and their provisioning.
- **Single Sign-On (SSO):** Configure identity providers and manage user access via Keycloak and

OAuth2 Proxy.

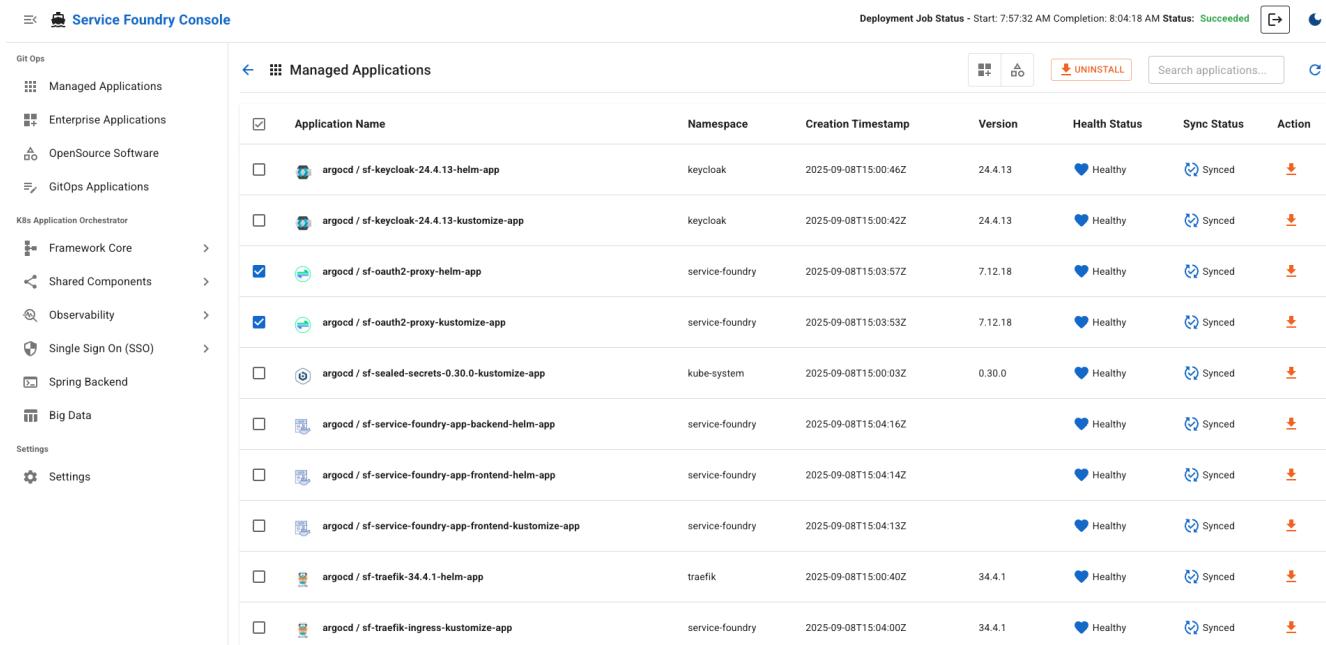
- **Big Data Stack:** Deploy scalable analytics infrastructure — including Apache Spark, Airflow, Neo4j, and OpenSearch.
- **Spring Backend Stack:** Easily deploy and operate Spring Boot-based microservices using standard templates and Helm charts.

Console Settings

- **Global Configuration:** Manage environment-wide settings related to GitOps, authentication, UI preferences, and more.

Managed Applications

The Managed Applications section provides a centralized dashboard for all Argo CD applications deployed in the Kubernetes cluster. From here, users can monitor, modify, and manage applications through a GitOps-driven workflow.



The screenshot shows the Service Foundry Console interface. On the left, there's a sidebar with categories like Git Ops, K8s Application Orchestrator, and Settings. The main area is titled "Managed Applications" and lists various Argo CD applications. Each application entry includes the name, namespace, creation timestamp, version, health status (with a heart icon), sync status (with a circular arrow icon), and an "Action" button. Some applications have checkboxes next to them. At the top right, there's a message about a deployment job status and a "UNINSTALL" button.

Managed Applications							
	Application Name	Namespace	Creation Timestamp	Version	Health Status	Sync Status	Action
<input type="checkbox"/>	argocd / sf-keycloak-24.4.13-helm-app	keycloak	2025-09-08T15:00:46Z	24.4.13	Healthy	Synced	
<input type="checkbox"/>	argocd / sf-keycloak-24.4.13-kustomize-app	keycloak	2025-09-08T15:00:42Z	24.4.13	Healthy	Synced	
<input checked="" type="checkbox"/>	argocd / sf-oauth2-proxy-helm-app	service-foundry	2025-09-08T15:03:57Z	7.12.18	Healthy	Synced	
<input checked="" type="checkbox"/>	argocd / sf-oauth2-proxy-kustomize-app	service-foundry	2025-09-08T15:03:53Z	7.12.18	Healthy	Synced	
<input type="checkbox"/>	argocd / sf-sealed-secrets-0.30.0-kustomize-app	kube-system	2025-09-08T15:00:03Z	0.30.0	Healthy	Synced	
<input type="checkbox"/>	argocd / sf-service-foundry-app-backend-helm-app	service-foundry	2025-09-08T15:04:16Z		Healthy	Synced	
<input type="checkbox"/>	argocd / sf-service-foundry-app-frontend-helm-app	service-foundry	2025-09-08T15:04:14Z		Healthy	Synced	
<input type="checkbox"/>	argocd / sf-service-foundry-app-frontend-kustomize-app	service-foundry	2025-09-08T15:04:13Z		Healthy	Synced	
<input type="checkbox"/>	argocd / sf-traefik-34.4.1-helm-app	traefik	2025-09-08T15:00:40Z	34.4.1	Healthy	Synced	
<input type="checkbox"/>	argocd / sf-traefik-ingress-kustomize-app	service-foundry	2025-09-08T15:04:00Z	34.4.1	Healthy	Synced	

Figure 8. Managed Applications

Platform users can:

- View the sync and health status of all deployed Argo CD applications
- Uninstall selected applications directly from the UI
- Filter between Open Source and Enterprise applications
- Navigate to application-specific views for further inspection

Application Filters and Actions

Table 2. Buttons on the header section:

Button	Action
	Toggle to display only Enterprise Applications .
	Toggle to display only Open Source Software applications .
Uninstall	Click to uninstall the selected applications from the cluster and remove it from the Git repository.

Table 3. Buttons on each application row:

Button	Action
Uninstall	Uninstall the application in a row.

Clicking on any application name opens the detailed application view with the following tabs:

Application Files

Users can inspect and modify manifest files directly in the browser-based editor. The Console supports a Git-aware editing workflow:

```

1  namespace: keycloak
2
3  resources:
4    - keycloak-namespace-24.4.13.yaml
5    - keycloak-credentials-secret-24.4.13.yaml
6    - keycloak-postgresql-credentials-secret-24.4.13.yaml

```

Figure 9. Application File Editor Tab

Table 4. Buttons on the editor header section:

Button	Action
	Hide File Tree.
	Show File Tree.
	Refresh the file tree to reflect the current state from Git.
	Enable editing mode for the selected manifest file.
	Discard edits and revert to the last committed version.
	Save changes made to the manifest files. This will stage the changes for commit.

Button	Action
	Discard all unsaved changes across files.
	Commit and push changes to the Git repository (triggers Argo CD sync).
	Add a Git commit message before pushing.
	Decrease font size in the file editor.
	Increase font size in the file editor.

Application Details Tab

The **Details** tab shows metadata for the selected Argo CD application, including:

- Application name and namespace
- Argo CD project
- Sync status (Synced / OutOfSync)
- Health status (Healthy / Degraded / Missing)
- Source repository path and revision

The screenshot shows the Service Foundry Console interface. At the top, there's a navigation bar with icons for Home, Projects, Applications, and a search bar. Below the navigation is a header bar with the text "Deployment Job Status - Start: 7:57:32 AM Completion: 8:04:18 AM Status: Succeeded" and a refresh button. The main content area has a sidebar on the left with categories like Git Ops, K8s Application Orchestrator, Settings, and more. The main panel displays the "argocd/sf-keycloak-24.4.13-kustomize-app" application details. The "DETAILS" tab is selected, showing fields such as Name (sf-keycloak-24.4.13-kustomize-app), Namespace (argocd), Health Status (Healthy), Sync Status (Synced), Project (service-foundry), Source Repo URL (git@github.com:nsalexamy/service-foundry-argocd.git), Target Revision (HEAD), Path (sf-apps/keycloak/sf-keycloak-24.4.13-kustomize-app/), Destination Name (in-cluster), Destination Namespace (keycloak), Sync Policy (Automated Prune: No, Automated Self Heal: No), Sync Options (CreateNamespace=true), and Labels (sf-app-type: kustomize, sf-component: keycloak, sf-module: infra, sf-version: 24.4.13, ui-category: core). There are also back and forward navigation arrows at the top of the main panel.

Figure 10. Application Details Tab

Kubernetes Resources Tab

The Resources tab lists all Kubernetes resources associated with the application. It allows users to:

- View resource type, name, namespace, and status
- Drill down into specific workloads (e.g., Deployments, Services, Secrets)

- Monitor resource state and lifecycle

The screenshot shows the Service Foundry Console interface. On the left, there's a sidebar with categories like Git Ops, K8s Application Orchestrator, and Settings. The main area is titled 'argocd / sf-keycloak-24.4.13-kustomize-app' and has tabs for APPLICATION FILES, DETAILS, and RESOURCES. The RESOURCES tab is active, showing a table with columns Group, Name, Kind, Namespace, Status, and Version. The table contains three rows: keycloak (Namespace, Synced, v1), keycloak-credentials (SealedSecret, keycloak, Synced, v1alpha1), and keycloak-postgresql-crede... (SealedSecret, keycloak, Synced, v1alpha1). At the bottom right of the table, there are buttons for 'Rows per page' (10), '1-3 of 3', and navigation arrows.

Figure 11. Kubernetes Resources Tab

Enterprise Applications

The Enterprise Applications section enables teams to define, deploy, and manage internal business applications sourced from private container registries. Service Foundry supports both Helm-based and Kustomize-based GitOps workflows, allowing users to scaffold applications using reusable templates and manage deployments through Argo CD.

The screenshot shows the Service Foundry Console interface. The sidebar on the left includes sections for Managed Applications, Enterprise Applications (which is currently selected), OpenSource Software, GitOps Applications, K8s Application Orchestrator, and Settings. The main area is titled 'Enterprise Applications' and features a table with columns: Argo CD App, Application Name, Namespace, Installed, Image Registry, Image Repository, Image Tag, Health Status, and Sync Status. There is a blue button '+ ADD NEW APPLICATION' at the top right of the table area. The table contains one row of data.

Figure 12. Enterprise Applications

To create a new application, click “Add New Application”, then select either a Helm or Kustomize deployment model depending on your application packaging format.

Create Enterprise Application

Users can scaffold enterprise applications by filling in a guided form. Service Foundry generates the required manifest files and Argo CD Application resources, commits them to the GitOps repository, and deploys them automatically.

The screenshot shows the Service Foundry Console interface for creating a new enterprise application. The left sidebar contains various project categories. The main area is titled 'Add New Enterprise Application' and includes sections for 'Common Information' and 'Kustomize Settings'. In 'Common Information', fields are filled with values like 'prj1' for Project Code and 'otel-spring-example' for Application Name. The 'Kustomize Settings' section displays a 'kustomization.yaml' file with specific resource definitions. At the top right, a deployment job status is shown as 'Succeeded'.

Figure 13. Create Enterprise Application

Common Fields

These fields are shared between Helm and Kustomize application types and are used to populate Kubernetes manifests and Argo CD configuration files.

Table 5. Common Fields Form

Field name	Description	example
Project Code	Logical identifier used as a prefix for Kubernetes resources.	prj1
Application Name	Application-specific identifier used in resource naming.	myapp
Namespace	Target Kubernetes namespace for deployment.	prj1
Version	Chart version (for Helm) or application version tag.	0.1.0
Image Registry	Hostname of the Docker image registry.	ghcr.io
Image Repository	Repository path to the container image.	o11y-otel-spring-example
Image Tag	Docker image tag to be deployed.	0.1.0
Replica Count	Number of application pods to deploy.	2
Container Port	Port the application container listens on.	8080

Field name	Description	example
Service Type	Kubernetes Service type. Options: ClusterIP, NodePort, LoadBalancer.	ClusterIP

Kustomize-Based Application

For applications defined using raw Kubernetes manifests, Service Foundry provides a visual resource composer for generating Kustomize-based applications. Users can dynamically add/remove/rename Kubernetes resource templates (e.g., Deployment, Service, ConfigMap, Secret, Ingress, etc.).

Table 6. Manage Kustomize Resources

Button	Action
Add Resource	Insert a new resource (e.g., Deployment, Service) into the manifest set.
Remove	Delete the selected resource from the Kustomize structure.
Rename	Rename an existing resource before creation.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: otel-spring-example
  labels:
    provider: service-foundry
spec:
  replicas: 1
  selector:
    matchLabels:
      app: otel-spring-example
  template:
    metadata:
      labels:
        app: otel-spring-example
    spec:
      containers:
        - name: otel-spring-example
          image: 445567090745.dkr.ecr.ca-central-1.amazonaws.com/o
            imagePullPolicy: Always
          ports:
            - containerPort: 8080
          envFrom:
            - configMapRef:
                name: my-config
          env:
            - name: JAVA_OPTS
              value: "-XX:+UnlockExperimentalVMOptions -XX:+UseCG

```

Figure 14. Create Kustomize Application - Kustomize

Once resources are configured, users can review and edit the generated YAML files in the built-in editor.

Deploy via GitOps

Click “Create Application” to generate the necessary manifest and Argo CD application files. These files are automatically committed to the GitOps repository and Argo CD will detect and deploy the new application to the Kubernetes cluster.

The screenshot shows a GitHub repository interface for a project named 'service-foundry-argocd'. The repository path is 'service-foundry-argocd/prj1-apps/otel-spring-example/prj1-otel-spring-example-0.1.0-kustomize-app'. A pull request titled 'nsa2 Add enterprise app otel-spring-example' has been merged, indicated by a green checkmark icon and the commit message 'Add enterprise app otel-spring-example'. The commit was made 2 minutes ago. The repository contains several files: 'kustomization.yaml', 'otel-spring-example-configmap.yaml', 'otel-spring-example-deployment.yaml', 'otel-spring-example-secret.yaml', and 'otel-spring-example-service.yaml'. All files were last committed 2 minutes ago.

Figure 15. GitOps Repository after Creating Kustomize Application

The newly created application will also appear in the **Managed Applications** section of the Console.

The screenshot shows the Service Foundry Console under the 'Enterprise Applications' section. A deployment job status bar at the top indicates 'Deployment Job Status - Start: 7:57:32 AM Completion: 8:04:18 AM Status: Succeeded'. The main table lists the application 'prj1-otel-spring-example-0.1.0-kustomize-app' with the following details:

Argo CD App	Application Name	Namespace	Installed	Image Registry	Image Repository	Image Tag	Health Status	Sync Status
prj1-otel-spring-example-0.1.0-kustomize-app	otel-spring-example	default	✓	445567090745.dkr.ecr.ca-central-1.amazonaws.com	o11y-otel-spring-example	0.1.0	Heartbeat Healthy	Synced

Figure 16. Enterprise Application Created

You can click on the application name to:

- Inspect deployment manifests
- View sync and health status
- Access related Kubernetes resources

Figure 17. Enterprise Application Details

All generated manifest files remain fully editable from the Console. Once modified, changes can be committed and pushed directly to the Git repository, triggering Argo CD to re-sync the application and apply updates to the cluster.

Open Source Software

Service Foundry provides a curated catalog of popular open-source applications that can be deployed seamlessly using Helm charts from public registries. These applications can be provisioned with just a few clicks and are fully integrated into the GitOps workflow.

Figure 18. Open Source Software Catalog

To install an application, select it from the catalog and click “Install”. Service Foundry guides you

through a streamlined setup process using Helm-based configuration templates.

Example: Installing PostgreSQL

When installing PostgreSQL, users are prompted to configure essential parameters such as:

- Database username and password
- Initial database name
- Helm chart version

The screenshot shows the Service Foundry Console interface. On the left, there's a sidebar with categories like Git Ops, Managed Applications, Enterprise Applications, OpenSource Software, GitOps Applications, K8s Application Orchestrator, Framework Core, Shared Components, Observability, Single Sign On (SSO), Spring Backend, Big Data, and Settings. The main area is titled 'Install PostgreSQL' under 'PostgreSQL'. It includes fields for 'Project Name' (set to 'oss'), 'Helm Chart Name' (set to 'postgresql'), 'Repo URL' (set to 'https://charts.bitnami.com/bitnami'), 'Chart Version' (set to '16.7.27'), 'Namespace' (set to 'opensource'), 'Release Name' (set to 'postgres'), 'Replica Count' (set to '1'), 'Image Registry' (empty), 'Image Repository' (empty), 'Image Tag' (empty), and 'Image Pull Policy' (set to 'IfNotPresent'). To the right, there's a large code editor window titled 'Observability Foundry Configuration' containing the following YAML code:

```
useArgocd: true
useArgoWorkflows: false
argocdAppPrefix: oss-
argocd:
  namespace: argocd
  project: service-foundry
  gitOpsRepolr: git@github.com:salexamy/service-foundry-argocd.git
  gitOpsRepoName: service-foundry-argocd
  gitOpsSshKeyPath: /opt/nsa/github-ssh/argocd_id_rsa
  gitOpsUserName: ns2-argocd
  gitOpsUserEmail: devops@company.com
common:
  envs: []
  postgresql:
    chartName: postgresql
    repoUrl: https://charts.bitnami.com/bitnami
    version: 16.7.27
    namespace: opensource
    releaseName: postgres
    replicaCount: 1
    imageRegistry: ""
    imageRepository: ""
    imageTag: ""
    imagePullPolicy: IfNotPresent
    serviceType: ClusterIP
    username: postgres
    password: postgres
    database: mydatabase
    postgresPassword: postgres
    replicationPassword: replication
    allowedSources:
      - 10.0.0.0/8
```

Figure 19. Install Open Source Software Application

After customization, clicking “Install Application” will:

- Generate Helm-based Kubernetes manifests
- Commit them to the GitOps repository
- Create a corresponding Argo CD application to manage the deployment

During installation, the Job Status is displayed in the header area to track progress in real time.

The screenshot shows the Service Foundry Console interface with the header 'Deployment Job Status - Start: 4:30:28 PM Status: In Progress'. This indicates that the deployment job is currently in progress.

Figure 20. Job Status - In Progress

GitOps Repository Integration

Once installation is complete, all manifests and configuration files are stored in the GitOps repository under a versioned path.

Name	Last commit message	Last commit date
...		
oss-postgres-16.7.27-helm-app/postgresql	Update ArgoCD applications from sso-foundry generator	7 minutes ago
oss-postgres-16.7.27-kustomize-app	Update ArgoCD applications from sso-foundry generator	7 minutes ago
oss-postgres-16.7.27-helm-app.yaml	Update ArgoCD applications from sso-foundry generator	7 minutes ago
oss-postgres-16.7.27-kustomize-app.yaml	Update ArgoCD applications from sso-foundry generator	7 minutes ago

Figure 21. GitOps Repository after Installing Open Source Software Application

The deployed application will also appear in the Managed Applications section of the Console, alongside other enterprise or custom workloads.

Application Details & Management

Click on any deployed open-source application to view detailed information such as:

- Kubernetes manifests
- Helm release values
- Resource status and health
- Argo CD sync history

Service Foundry Console
Deployment Job Status - Start: 4:41:52 PM Completion: 4:42:44 PM Status: Succeeded

Git Ops

Managed Applications
Enterprise Applications
OpenSource Software
GitOps Applications

K8s Application Orchestrator

Framework Core
Shared Components
Observability
Single Sign On (SSO)
Spring Backend
Big Data

Settings

Settings

Learn More (OSS) - PostgreSQL

Namespace `opensource`

Helm App
Kustomize App

UNINSTALL

Namespace `prj1`

Helm App
Kustomize App

UNINSTALL

Figure 22. Open Source Software Application Details

Applications can be deployed into different namespaces, and each instance is managed as an isolated Argo CD application with its own configuration and lifecycle.

Application Actions

Buttons to manage the application:

Table 7. Application Actions

Button	Action
Helm App	View Helm-specific values and manifests associated with the deployment.
Kustomize App	View Kustomize manifests if the application was scaffolded using Kustomize.
UNINSTALL	Remove the application from the Kubernetes cluster and delete the associated Argo CD application. Note: Manifest files will remain in the Git repository for audit purposes.

GitOps Applications

The GitOps Applications section enables full lifecycle management of raw Kubernetes manifests stored in the Git repository. Users can create, edit, deploy, and remove applications using a Git-centric workflow—without needing to leave the console.

The screenshot shows the Service Foundry Console interface. On the left, there's a sidebar with navigation links like 'Managed Applications', 'Enterprise Applications', 'OpenSource Software', 'GitOps Applications' (which is currently selected), 'Framework Core', 'Shared Components', 'Observability', 'Single Sign On (SSO)', 'Spring Backend', 'Big Data', and 'Settings'. The main area is titled 'GitOps Applications' and shows a list of applications. At the top right, it says 'Deployment Job Status - Start: 4:41:52 PM Completion: 4:42:44 PM Status: Succeeded'. Below that, there are buttons for 'DELETE APPS', 'INSTALL', 'UNINSTALL', and 'Search applications...'. The application list table has columns for 'Namespace', 'Version', and 'Actions'. The table contains several entries, each with a green checkmark icon and a small icon representing the application type (e.g., PostgreSQL, Keycloak). The actions column includes icons for edit, delete, and other operations.

Figure 23. GitOps Applications

Service Foundry allows you to reuse existing Kubernetes manifests in your GitOps repository to scaffold new Argo CD applications.

Application List — Actions

Each application entry supports quick access to common actions:

Table 8. Action Buttons in the application list

Button	Action
	Copy the full path to the Argo CD application file for reference or external tooling.
	Edit the Argo CD application manifest directly in the built-in file editor.
	Deploy the application to the cluster by creating the corresponding Argo CD application. Available only if the application is not yet installed.
	Uninstall the application from the cluster and remove all associated Kubernetes resources. Available only if already installed.
	Permanently delete the application manifest from the Git repository. This action cannot be undone.

Batch Operations — Header Actions

The header also provides multi-application controls for bulk operations:

Table 9. Header Buttons for Batch Operations

Button	Description
 DELETE APPS	Delete selected GitOps application manifests from the repository. Irreversible.
 INSTALL APPS	Create Argo CD applications for selected manifests and deploy them to the cluster.
 UNINSTALL APPS	Uninstall selected applications and remove their Kubernetes resources from the cluster.

Click an application name to view detailed information and edit files in place.

View GitOps Application

Users can drill down into each GitOps application to inspect and manage its manifests. This includes editing files, creating Argo CD apps, and managing installation status.

```

1
2  apiVersion: argoproj.io/v1alpha1
3  kind: Application
4  metadata:
5    name: prj1-postgres-16.7.27-helm-app
6    namespace: argocd
7  labels:
8    sf-module: "opensource"
9    sf-component: "postgresql"
10   sf-app-type: "helm"
11   sf-version: "16.7.27"
12   ui-category: "oss"
13   #sf-app-path: "prj1-apps/postgresql/prj1-postgres-16.7.27-helm-app.yaml"
14   finalizers:
15     - resources-finalizer.argocd.argoproj.io
16   annotations:
17     argocd.argoproj.io/sync-wave: "1"
18   spec:
19     destination:
20       namespace: prj1
21       #server: https://kubernetes.default.svc
22       name: in-cluster
23     project: service-foundry
24     syncPolicy:
25       automated: {}
26       syncOptions:
27         - CreateNamespace=true
28     source:
29       renURL: git@github.com:calevamu/service-foundry-arnord.git

```

Figure 24. GitOps Application Details

Following buttons are available in the header for application-specific actions:

Table 10. Header Buttons for Application Management

Button	Action
View App Files	Open the file tree to browse and edit application manifests.
Create ArgoCD App	Create a new Argo CD application to deploy the manifest to the Kubernetes cluster.
Uninstall ArgoCD App	Remove the Argo CD application and associated resources from the cluster.
DELETE APP	Permanently delete the application manifest from the Git repository. This action is irreversible.

Any edits to the manifest files can be committed directly from the console, triggering Argo CD to sync changes and apply them to the cluster automatically.

Kubernetes Stack Orchestration

Service Foundry supports modular, stack-based orchestration for managing complex Kubernetes workloads. Each stack bundles a set of components into a cohesive deployment unit, enabling teams to layer functionalities in a controlled and scalable manner.

Users can selectively deploy the stacks they need, in any order, while respecting inter-stack dependencies. For example, the Observability Stack can be layered on top of the Framework Core and Shared Components.

Available Stacks:

- Framework Core

- Shared Components
- Observability Stack
- Single Sign-On (SSO) Stack
- Spring Backend Stack (*Work In Progress*)
- Big Data Stack (*Work In Progress*)

Framework Core

The **Framework Core** is the foundation of Service Foundry. It includes critical services required for the platform to operate reliably from initial setup onward.

The screenshot shows the Service Foundry Console interface. On the left, there's a sidebar with navigation links for Git Ops (Managed Applications, Enterprise Applications, OpenSource Software, GitOps Applications), Kubernetes Stack Orchestrator (Framework Core, Component Gallery, Shared Components, Observability, Single Sign On (SSO), Spring Backend, Big Data), and Settings. The 'Component Gallery' link under 'Framework Core' is highlighted. The main area is titled 'Components Gallery' and displays several components:

- ArgoCD**: Shows HELM, KUSTOMIZE, and DELETE options.
- Sealed-secrets**: Shows HELM, KUSTOMIZE, and DELETE options.
- Keycloak**: Shows HELM, KUSTOMIZE, and DELETE options.
- Traefik**: Shows HELM, KUSTOMIZE, and DELETE options.
- Service-foundry-app-frontend**: Shows HELM, KUSTOMIZE, and DELETE options.
- Service-foundry-app-backend**: Shows HELM, KUSTOMIZE, and DELETE options.

 Each component card has a green checkmark icon indicating it is preconfigured. At the top right of the main area, there's a status bar showing 'Deployment Job Status - Start: 4:41:52 PM Completion: 4:42:44 PM Status: Succeeded'.

Figure 25. Framework Core Components

Each component in this stack is preconfigured for seamless integration. Users may review configuration details and adjust settings as needed—but it is not recommended to uninstall any components from this stack, as they are essential for overall platform stability and functionality.

Shared Components

The Shared Components stack includes reusable services and Kubernetes Operators that provide cross-cutting functionality across multiple application domains and stacks.

The screenshot shows the Service Foundry Console interface. On the left, there's a sidebar with navigation links for Git Ops, K8s Application Orchestrator, and Settings. Under Component Gallery, 'Shared Components' is selected. The main area is titled 'Components Gallery' and lists four operators: Cert-manager, Prometheus-operator, Otel-operator, and Spark-operator. Each operator card includes icons for Helm, Kustomize, and Delete.

Figure 26. Shared Components

Operators such as prometheus-operator, opentelemetry-operator, and spark-operator are part of this stack. These are not mandatory, and users can choose to install only the components relevant to their use case.

For example, if you're deploying the Observability Stack, it's recommended to install the prometheus-operator and opentelemetry-operator beforehand.

Click **Orchestrate** to generate Kubernetes manifests, commit them to the GitOps repository, and deploy them via Argo CD.

The screenshot shows the Service Foundry Console interface. The sidebar includes 'Deploy Components' under 'Shared Components'. The main area is titled 'Shared Components Configuration' and shows the configuration for the 'OpenTelemetry Operator'. On the left, the configuration file 'infra-foundry-config.yaml' is displayed with sections for 'Common', 'ArgoCD Configuration', and 'OpenTelemetry Operator'. The 'OpenTelemetry Operator' section is expanded, showing 'Enabled (Install)' with 'Namespace' set to 'opentelemetry-operator-system', 'Name' set to 'otel-operator', 'Version' set to '0.127.0', and 'Replica Count' set to '1'. On the right, the generated 'Infra Foundry Configuration' is shown as a code editor with syntax highlighting for YAML. The code defines ArgoCD application prefixes for 'infra-' and 'cert-' and specifies configurations for the 'otel-operator' and 'prometheus-operator'.

```

useArgocd: true
useArgoWorkflows: false
argocdAppPrefix: infra-
common:
  cloudProvider: aws
  containerRegistry: ""
argocd:
  namespace: argocd
  project: service-foundry
gitopsRepoUrl: git@github.com:nsalexamy/service-foundry-argocd.git
gitopsRepoName: service-foundry-argocd
gitopsSshKeyPath: /opt/nsa2/github-ssh/argocd_id_rsa
gitopsUserName: ns2
gitopsUserEmail: ns2@example.com
certManager:
  enabled: true
  namespace: cert-manager
  chartName: cert-manager
  releaseName: cert-manager
  version: v1.17.2
  replicaCount: 1
  repourl: https://charts.jetstack.io
  needTlsAlt: true
prometheusOperator:
  enabled: true
  namespace: default
  releaseName: prometheus-operator
  chartName: prometheus-operator
  version: v0.83.0
  replicaCount: 1
otelOperator:
  enabled: true
  namespace: opentelemetry-operator-system
  releaseName: otel-operator
  chartName: opentelemetry-operator
  version: 0.127.0

```

Figure 27. Deploy Shared Components

Applicable Domain

Table 11. Supported Domains and Required Components

Domain	Required Components
Observability	Cert-manager, Prometheus-operator, Opentelemetry-operator
Big Data	Cert-manager, Spark-operator

Upon deployment, manifests are organized under the infra-apps directory in the GitOps repository:

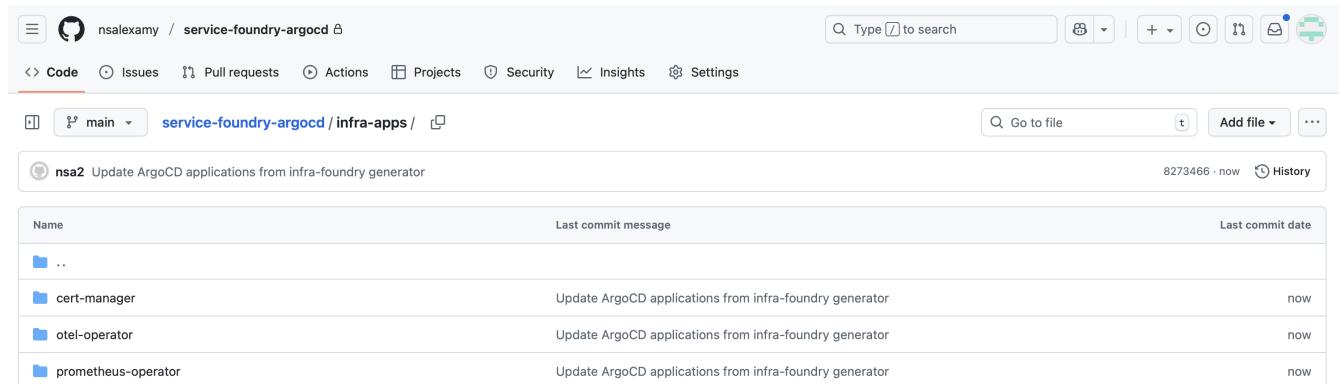


Figure 28. GitOps Repository - Shared Components

The stack will also appear in the **Managed Applications** section of the console:

The screenshot shows the Service Foundry Console interface. On the left, there is a sidebar with navigation links for 'Git Ops', 'Enterprise Applications', 'OpenSource Software', 'GitOps Applications', 'K8s Application Orchestrator', 'Framework Core', 'Shared Components', 'Observability', 'Single Sign On (SSO)', 'Spring Backend', and 'Big Data'. The main area is titled 'Managed Applications' and displays a table with three rows of data. The columns are 'Application Name', 'Namespace', 'Creation Timestamp', 'Version', 'Health Status', 'Sync Status', and 'Action'. The first row corresponds to the 'cert-manager' folder in the GitHub repository, the second to 'otel-operator', and the third to 'prometheus-operator'. All applications are marked as 'Healthy' and 'Synced'.

Figure 29. Managed Applications - Shared Components

Observability Stack

The **Observability Stack** provides full support for logs, metrics, and traces. It is designed to adapt to different environments—**Development**, **Staging**, and **Production**—by offering tailored profiles for each.

The screenshot shows the Service Foundry Console's Components Gallery. On the left sidebar, under the 'Component Gallery' section, there is a link to 'Deploy Components'. The main area displays a grid of 16 components, each with a icon, name, and deployment options (HELM, KUSTOMIZE, DELETE). The components are categorized as follows:

- Row 1:** Cassandra, Data-prepper, Grafana, Jaeger
- Row 2:** Kubelet-cadvisor-collector, Loki, Mimir, Minio
- Row 3:** Opensearch, Opensearch-dashboards, Otel-collector, Otel-spring-example
- Row 4:** Prometheus, React-o11y-app, Tempo

Figure 30. Observability Stack

Component dependencies are considered during orchestration. For example, the Otel Collector may require services like Loki, Tempo, and Prometheus to be deployed together.

Click **Orchestrate** to start the deployment process.

The screenshot shows the Service Foundry Console's Observability Foundry Configuration screen. On the left sidebar, under the 'Deploy Components' section, there is a link to 'Observability'. The main area has two panes: a left pane for the configuration file and a right pane for the configuration editor.

Left Pane (Configuration File):

```
o11y-foundry-config.yaml
Argocd Application Prefix
o11y-
```

Right Pane (Editor):

Applicable Profiles: DEV STAGING PRODUCTION RESET **DEPLOY**

```
Observability Foundry Configuration
1 useArgocd: true
2 useArgoWorkflows: false
3 argocdAppPrefix: o11y-
4 common:
5   namespace: o11y
6   cloudProvider: aws
7   containerRegistry: ""
8   rootDomain: nsaz2.com
9   envs: []
10 argocd:
11   namespace: argocd
12   project: service-foundry
13   gitOpsRepoUrl: git@github.com:nsalexam/service-foundry-argocd.git
14   gitOpsRepoName: service-foundry-argocd
15   gitOpsSshKeyPath: /opt/nsaz2/github-ssh/argocd_id_rsa
16   gitOpsUserName: nsaz2
17   gitOpsUserEmail: nsaz2@example.com
18 eauth2:
19   oidcIssuerUrl: ""
20   clientId: nsaz2
21   clientSecret: gZ343TCd0kBehqT0kGZFkbL4WvXoa3Ss
22   serverType: keycloak
23   keycloakNamespace: keycloak
24   keycloakServiceName: keycloak
25   keycloakRealm: default
26   keycloakServiceType: LoadBalancer
27   keycloakProtocol: http
28 prometheus:
29   enabled: true
30   releaseName: prometheus
31   chartName: prometheus
32   version: 0.83.0
33   replicaCount: 1
34   grafana:
35     enabled: true
36     chartName: grafana
```

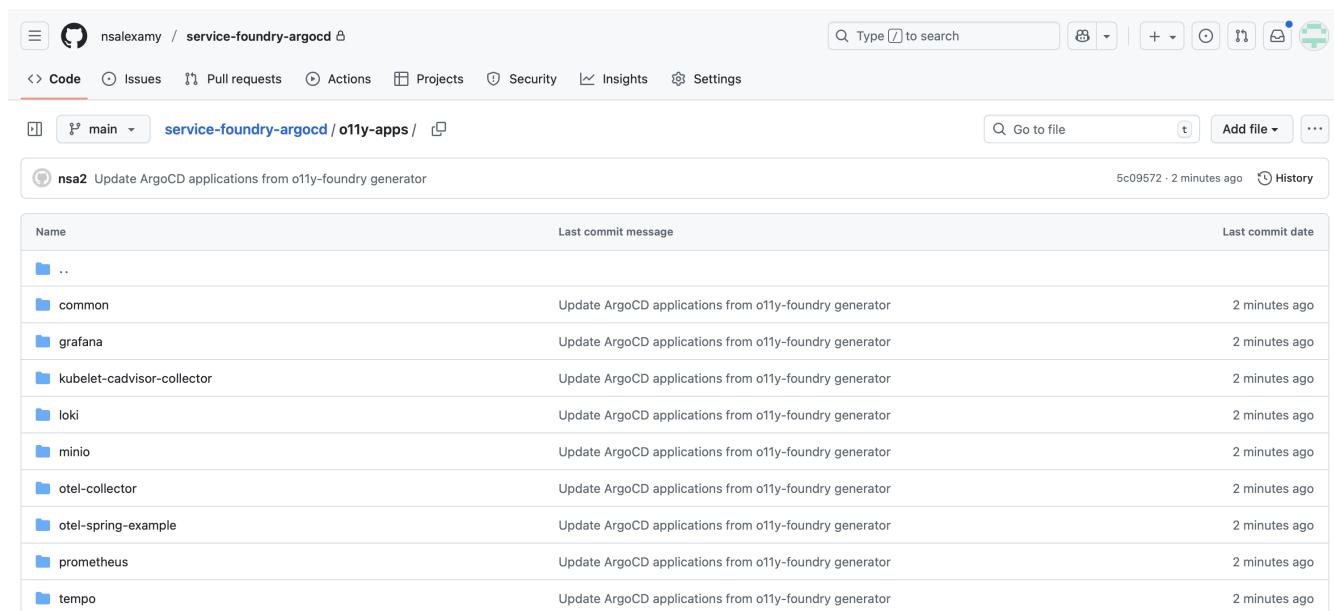
Figure 31. Deploy Observability Stack

Available Profiles

Table 12. Observability Stack Profiles

Profile	Description
Dev Profile	A lightweight configuration including Prometheus, Grafana, Loki, and Otel Collector—ideal for development and local testing.
Staging Profile	Adds components like OpenSearch and S3-compatible object storage to support staging environments and persistent data retention.
Production Profile	A comprehensive observability suite including Jaeger, Cassandra, and high-availability configurations suitable for production workloads.

Each profile triggers manifest generation and GitOps deployment, with resources organized under the observability-apps directory:



Name	Last commit message	Last commit date
...		
common	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
grafana	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
kubelet-cadvisor-collector	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
loki	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
minio	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
otel-collector	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
otel-spring-example	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
prometheus	Update ArgoCD applications from o11y-foundry generator	2 minutes ago
tempo	Update ArgoCD applications from o11y-foundry generator	2 minutes ago

Figure 32. GitOps Repository - Observability Stack

Once deployed, the application appears in the Managed Applications section:

Figure 33. Managed Applications - Observability Stack

Single Sign-On (SSO) Stack

The **SSO Stack** enables authentication and secure access management across the platform. It integrates Keycloak, OAuth2 Proxy, and Traefik to provide seamless SSO for internal and third-party applications.

Figure 34. Single Sign-On (SSO) Stack

Traefik IngressRoute and OAuth2 Proxy are preconfigured to secure access to UIs such as Grafana

and the Service Foundry Console.

Deploying the SSO Stack

Click **Orchestrate** to deploy the SSO stack. This creates manifest files, commits them to Git, and provisions the stack via Argo CD.

The screenshot shows the Service Foundry Console interface. On the left, a sidebar lists various categories: Git Ops, Managed Applications, Enterprise Applications, OpenSource Software, GitOps Applications, Kubernetes Stack Orchestrator, Framework Core, Shared Components, Observability, Single Sign On (SSO), Component Gallery, Deploy Components (which is selected), Resource Servers, Spring Backend, Big Data, and Settings. The main area is titled "SSO Foundry Configuration" and displays a YAML file named "sso-foundry-config.yaml". The configuration includes sections for ArgoCD Application Prefix (set to "sso-"), Common, ArgoCD Configuration, OAuth2, Keycloak Reference, Traefik Ingress, and OAuth2 Proxy. The "OAuth2 Proxy" section is expanded, showing fields for Namespace (set to "o11y"), Helm Chart Name (set to "oauth2-proxy"), Helm Release Name (set to "oauth2-proxy"), Version (set to "7.12.18"), and Replica Count (set to "1"). To the right of the configuration editor is a "Deployment Job Status" card indicating "Start: 11:35:54 AM Completion: 11:39:29 AM Status: Succeeded". A "DEPLOY" button is located at the top right of the configuration editor.

```
useArgocd: true
useArgoWorkflows: false
argocdAppPrefix: sso-
common:
  namespace: o11y
  rootDomain: nsa2.com
  envs: []
argocd:
  namespace: argocd
  project: service-foundry
  gitOpsRepoUrl: git@github.com:nsalexamy/service-foundry-argocd.git
  gitOpsRepoName: service-foundry-argocd
  gitOpsSshKeyPath: /opt/nsa2/github-ssh/argocd_id_rsa
  gitOpsUserName: nsa2
  gitOpsUserEmail: nsa2@example.com
  oauth2:
    oidcIssuerUrl: ""
    clientId: nsa2
    clientSecret: gZ343TCd0kBehqT0KGZFkbL4WvXoa3Ss
    serverType: keycloak
    keycloakNamespace: keycloak
    keycloakServiceName: keycloak
    keycloakRealm: default
    keycloakServiceType: LoadBalancer
    keycloakProtocol: http
    keycloak:
      enabled: false
      namespace: keycloak
      chartName: keycloak
      releaseName: keycloak
      serviceType: LoadBalancer
      realm: default
      protocol: http
      traefikIngress:
        enabled: true
version: 2.8.1
```

Figure 35. Deploy SSO Stack

SSO Configuration

```

Deployment Job Status - Start: 11:35:54 AM Completion: 11:39:29 AM Status: Succeeded
45 name: o11y-sso-ingress
46 services:
47   - serviceName: grafana
48     portName: service
49       subdomain: grafana
50       namespace: o11y
51       namespace: o11y

```

Figure 36. SSO Configuration

Oauth2 Proxy Ingress Form

The IngressRoute configuration form allows you to define routing rules for SSO-protected applications. Key fields include:

Table 13. IngressRoute Configuration Fields

Field name	Description	Example
Name	Unique name for the Ingress resource	o11y-sso-ingress
Namespace	Kubernetes namespace	o11y
Service Name	Kubernetes service to route to	grafana
Port Name	Target port name on the service	service
Subdomain	Subdomain for routing	grafana

A subdomain like grafana with root domain nsa2.com will create a route: <http://grafana.nsa2.com>.

You can verify the deployed route:

```
$ kubectl -n o11y get ingressroutes o11y-sso-ingress-route -o yaml | jq '.spec'
```

Sample IngressRoute manifest

```

entryPoints:
  - web
routes:
  - kind: Rule

```

```

match: Host(`grafana.nsa2.com`)
middlewares:
  - name: cors-headers
  - name: forward-auth
services:
  - name: grafana
    port: service

```

Once deployed, the SSO application is visible in both the GitOps repository and the Service Foundry Console:

Name	Last commit message	Last commit date
..		
oauth2-proxy	Update ArgoCD applications from sso-foundry generator	4 minutes ago
traefik-ingress	Update ArgoCD applications from sso-foundry generator	4 minutes ago

Figure 37. GitOps Repository - SSO Stack

Application Name	Namespace	Creation Timestamp	Version	Health Status	Sync Status	Action
argocd / sso-oauth2-proxy-helm-app	o11y	2025-09-09T20:04:15Z	7.12.18	Healthy	Synced	
argocd / sso-oauth2-proxy-kustomize-app	o11y	2025-09-09T20:04:11Z	7.12.18	Healthy	Synced	
argocd / sso-traefik-ingress-kustomize-app	o11y	2025-09-09T20:04:18Z	34.4.1	Healthy	Synced	

Figure 38. Managed Applications - SSO Stack

Resource Servers

SSO-protected applications, such as Grafana, are listed under Resource Servers. These services inherit the same credentials and session state for unified access control.

Host	Namespace	Middlewares	Services	Ports
http://grafana.nsa2.com	o11y	cors-headers, forward-auth	grafana	service
http://sfapp.nsa2.com	service-foundry	cors-headers, forward-auth	service-foundry-app-frontend	http
http://sfapp-backend.nsa2.com	service-foundry	cors-headers, forward-auth	service-foundry-app-backend	http

Figure 39. SSO Resource Servers

When accessing Grafana, users are redirected to the Keycloak login page for authentication. After successful login, they are granted access to Grafana without needing to re-enter credentials.

Figure 40. Accessing Grafana with SSO

Spring Backend Stack (Work In Progress)

The upcoming **Spring Backend Stack** is designed to support enterprise-grade Java applications using Spring Boot. It will include runtime dependencies such as PostgreSQL, Redis, RabbitMQ, and configuration tools tailored for microservice development and deployment.

Big Data Stack (Work In Progress)

The **Big Data Stack** will enable scalable data processing using popular open-source technologies. It includes support for Apache Spark, Airflow, OpenSearch, Neo4j, MinIO, and dbt. This stack is intended for teams building data pipelines, graph analytics, or large-scale ETL workflows.