1. Backstage 관리하기

Backstage에 CloudForet 서비스를 Software Catalog의 Component로 등록하기

Component에 CloudForet 서비스 등록하기

(1) Backstage가 Component로 인식할 수 있는 yaml 파일 생성하기

- `backstage/component-info.yaml` 파일을 아래와 같이 작성해줍니다.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: cloudforet-inventory
  namespace: cloudforet-core
  annotations:
    backstage.io/techdocs-ref: dir:.
    backstage.io/managed-by-origin-location: url:https://github.com
/cloudforet-io/inventory/blob/master/backstage/component-info.yaml
spec:
  type: coreService
  lifecycle: experimental
  owner: bluese@mz.co.kr
```

- 문서를 작성합니다. 예를 들어, `backstage/docs/index.md` 에 아래와 같이 작성해봅니다.

```
### CloudForet Inventory Service

- This is CloudForet inventory service, registered as a Backstage
component
```

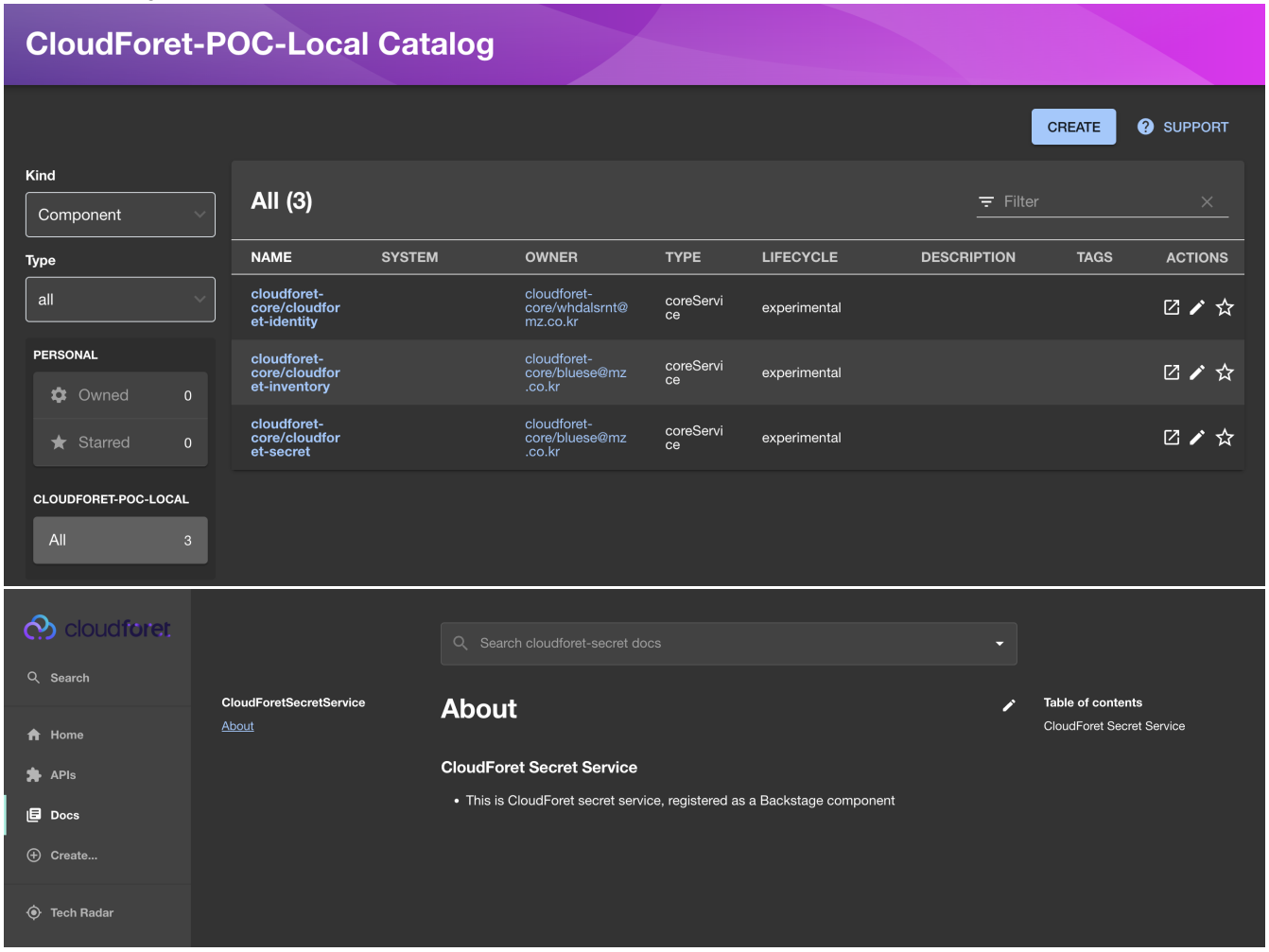
- Backstage TechDocs를 생성해주는 mkdocs를 위한 설정 파일인 `backstage/mkdocs.yml` 을 생성합니다.

```
site_name: CloudForetInventoryService
site_description: CloudForet Inventory Service
nav:
  - About: index.md
plugins:
  - techdocs-core
```

- 마지막으로 `app-config.yaml` (상황에 따라 `app-config.local.yaml` 또는 `app-config.production.yaml`) 의 `catalog.locations` 부분에 아래와 같이 `backstage/component-info.yaml` 을 가리키는 GitHub repository url을 추가합니다.

```
catalog:
  locations:
    - type: url
      target: https://github.com/cloudforet-io/identity/blob/master
/backstage/component-info.yaml
```

- 이제 Backstage 애플리케이션을 다시 구동하면, 아래와 같이 노출됩니다.



Software Template 등록하기

- Backstage Software Template은 개발자들이 기존에 존재하는 코드베이스(boilerplate, template)를 토대로 새로운 프로젝트를 빠르게 생성하고 시작하기 위한 기능입니다.
  GitHub의 template repository와 유사하다고 생각하시면 됩니다.
- Backstage에 Software Template으로 사용할 예시 프로젝트를 참고해주세요.

1. Software Template으로 사용할 프로젝트 생성

- Software template으로 활용할 프로젝트의 기본 구조는 아래와 같습니다.

```
|-- template.yaml
|-- skeleton
      |-- Dockerfile
      |-- main.go
      |-- go.mod
      |-- catalog-info.yaml
      |-- mkdocs.yml
      |-- docs
            |-- index.md
```

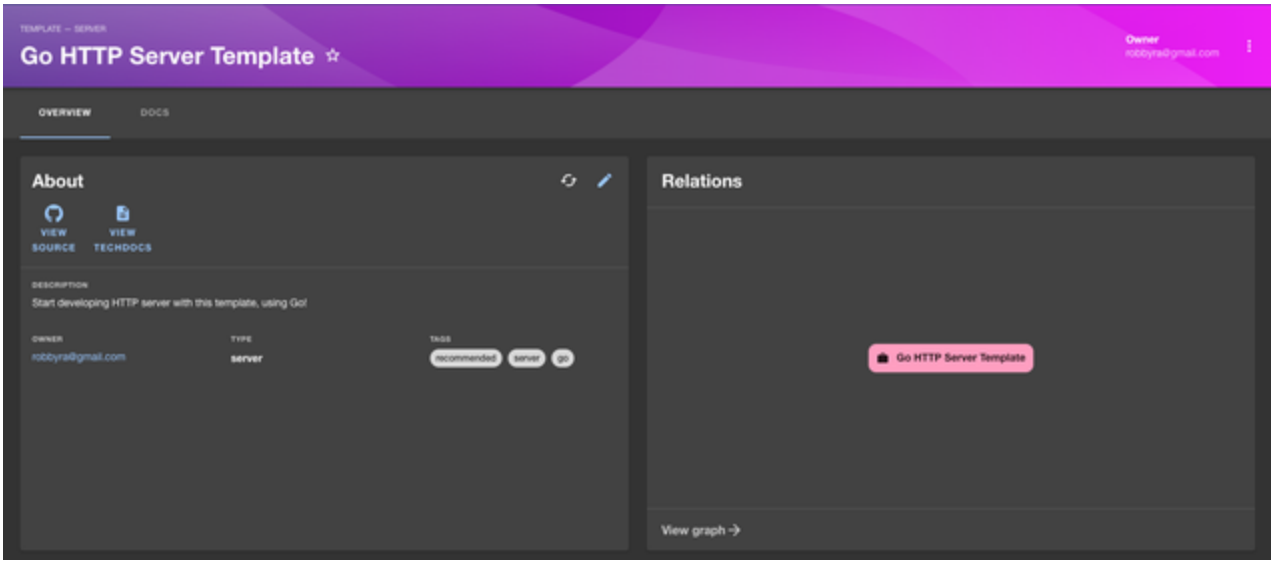
- `template.yaml` : 이 Software template 자체에 대한 설정 정보들을 담습니다.
  - Template project의 이름, 제목, maintainer 등
  - 사용자가 이 Software Template으로부터 프로젝트를 만들 때 필요한 단계 정의
    - 이 단계에서 사용자가 입력하는 값들을 `skeleton/catalog-info.yaml` , `skeleton/mkdocs.yml` 및 `skeleton/docs/index.md` 에서 사용할 수 있습니다.
- `skeleton` : 사용자가 이 Software Template으로부터 새로운 프로젝트를 만들 때 들어갈 파일들을 포함합니다.
- `skeleton/catalog-info.yaml` : 새롭게 생성한 프로젝트를 Backstage에서 인식하기 위한 설정 정보가 담긴 파일입니다.
- `skeleton/mkdocs.yml` : 문서 생성에 필요한 정보를 지정합니다.
- `skeleton/docs/index.md` : 문서를 정의합니다.

2. Software Template으로 등록

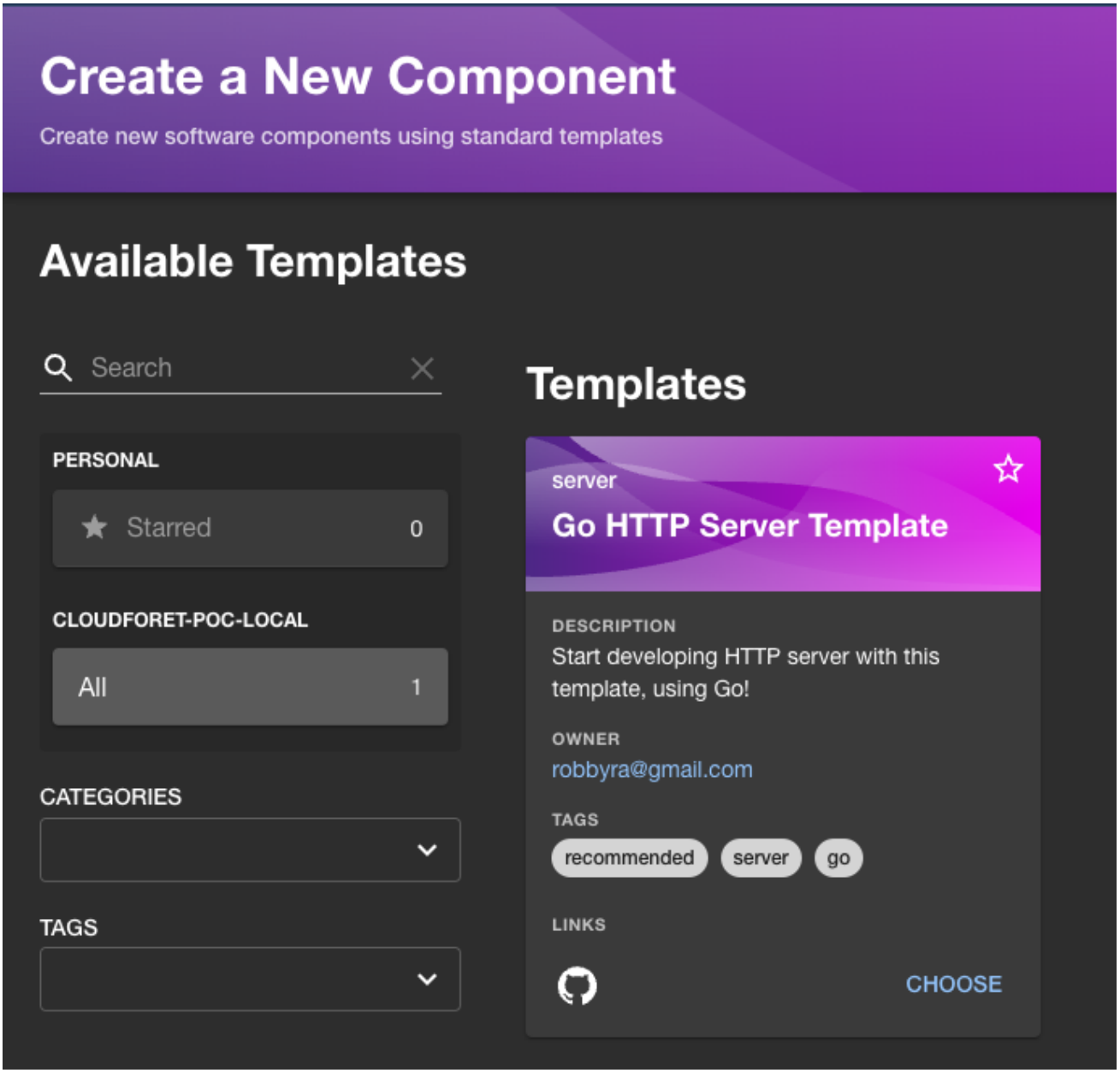
- Component를 등록할 때와 마찬가지로 아래와 같이 `app-config.yml` 에 추가합니다.

```
catalog:
  locations:
    #..
    - type: url
      target: https://github.com/sang-w0o/backstage-template-
example-repo/blob/main/template.yaml
      rules:
        - allow: [Template]
```

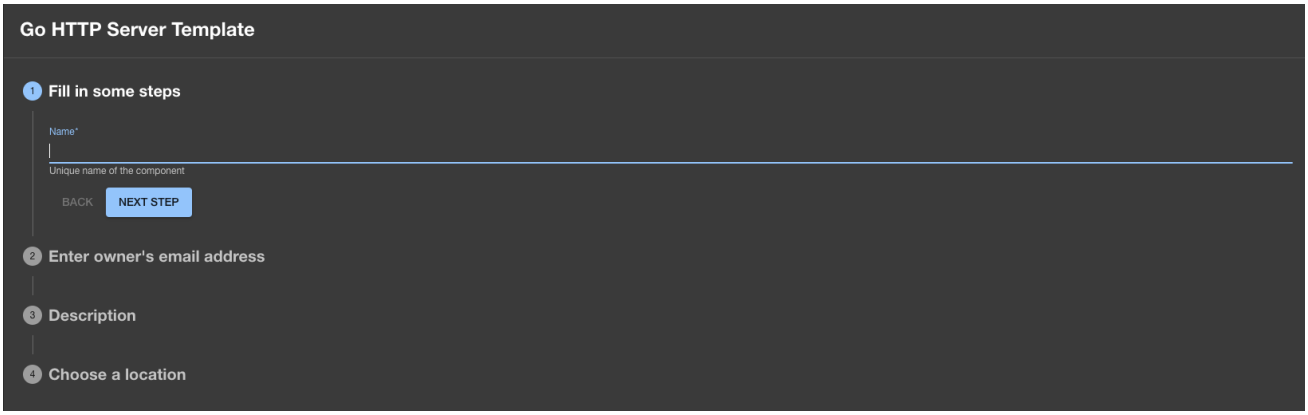
- 추가가 완료되면, 아래와 같이 Backstage에서 Software Template을 조회할 수 있습니다.



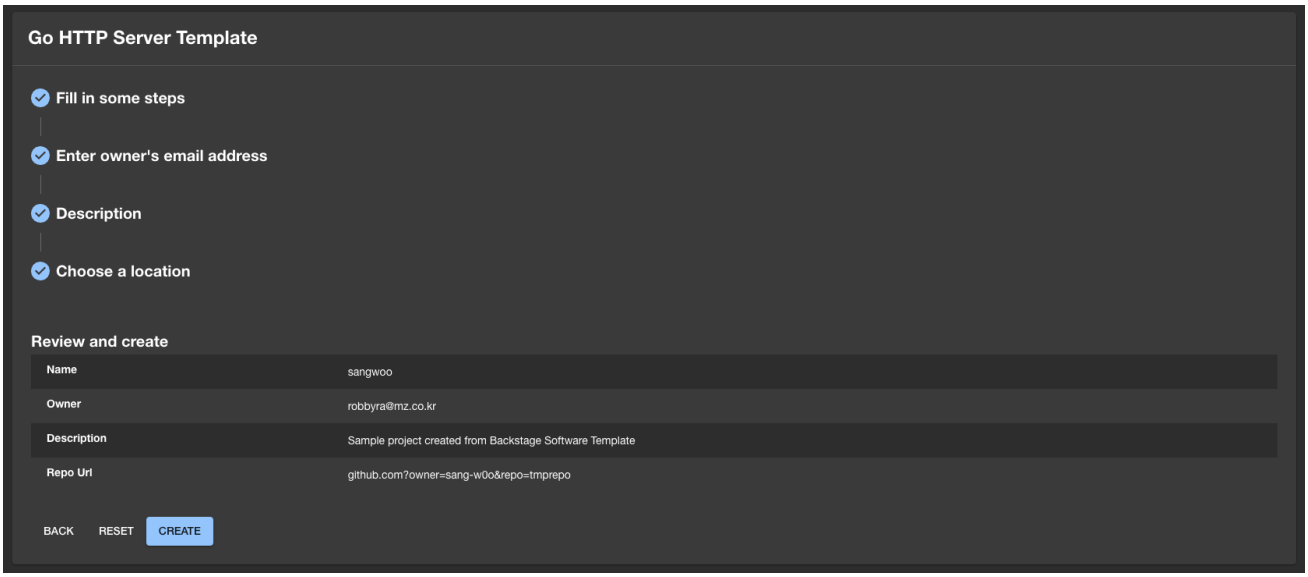
- 이후 `Create...` 페이지에서 이 template으로부터 새로운 프로젝트를 생성할 수 있습니다.



- 선택하고 넘어가면, template 프로젝트의 `template.yaml` 에 정의한 프로젝트를 생성하기 위한 단계들이 렌더링됩니다.



- 아래와 같이 입력했다고 가정해봅니다.
  - (1) Fill in some steps: sangwoo
  - (2) Enter owner's email address: robbyra@mz.co.kr
  - (3) Description: Sample project created from Backstage Software Template
  - (4) Choose a location:
    - owner: sang-w0o
    - repository: tmprepo
- 모든 정보를 입력하고 CREATE 버튼을 클릭합니다.



- 생성이 완료된 후 새로운 프로젝트의 코드 구조는 아래와 같습니다.

```
|-- Dockerfile
|-- main.go
|-- go.mod
|-- catalog-info.yaml
|-- mkdocs.yml
|-- docs
      |-- index.md
```

- 그리고 Backstage가 사용하는 `catalog-info.yaml`, `mkdocs.yml` , 그리고 `docs/index.md` 에는 위에서 프로젝트를 생성할 때 입력한 값들이 들어가 있습니다.
  예를 들어, template 레포지토리의 `skeleton/catalog-info.yaml` 파일은 아래와 같았습니다.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: ${{ values.name | dump }}
  namespace: server-components
  annotations:
    backstage.io/techdocs-ref: dir:.
spec:
  type: server
  lifecycle: experimental
  owner: ${{ values.owner | dump }}
```

- 그리고 새롭게 만들어진 프로젝트의 `catalog-info.yaml` 파일은 아래와 같습니다.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: "sangwoo"
  namespace: server-components
  annotations:
    backstage.io/techdocs-ref: dir:.
spec:
  type: server
  lifecycle: experimental
  owner: "robbyra@mz.co.kr"
```

- 이렇게 새롭게 생성된 프로젝트에는 사용자가 입력한 값들이 들어갈 수 있습니다.
  이후 이 프로젝트를 `app-config.yaml` 에 등록하면, 다른 component들과 마찬가지로 사용할 수 있습니다.

Component를 Kubernetes에 등록하기

- 공식 문서
- Backstage에서 이야기하는 Kubernetes는 클러스터 관리자가 아닌, 각 서비스의 담당자를 위해 만들어진 기능입니다.
  즉, 개발자들이 쉽게 자신이 배포한 서비스의 health check, 상태 등을 쉽게 파악하기 위함입니다.

(1) 프론트엔드 코드 설정해주기

- 공식 문서
- Backstage는 크게 두 가지의 컴포넌트로 이루어져 있습니다.
  - `app` : frontend
  - `backend` : backend
- app, backend는 각각 다양한 플러그인을 설치해 사용할 수 있습니다.
  여기서는 Kubernetes를 설정하기에 이와 관련된 플러그인을 설치하고, 설정해줄 것입니다.
- 우선 프론트엔드부터 설정합니다.
  - 터미널에 아래 명령을 입력해 `packages/app` 에 K8S plugin을 설치해줍니다.

```
yarn add --cwd packages/app @backstage/plugin-kubernetes
```

- 다음으로 `packages/app/src/components/catalog/EntityPage.tsx` 로 이동해 아래 내용을 작성합니다.

  여기서 공식 문서대로만 작성하면, 컴포넌트를 선택한 화면에서 Kubernetes 탭이 뜨지 않습니다.
  이 이유는 위의 `Backstage CloudForet  Software Catalog Component` 단계에서 `backstage/component-info.yaml` 에서 `spec.type` 을 coreService로 지정했기 때문입니다.
- 아래 코드에 적절한 주석을 달아놓았습니다.

```
// import
import { EntityKubernetesContent } from '@backstage/plugin-
kubernetes';

/**
 *  cloudForetCoreServiceEntityPage
 * serviceEntityPage, websiteEntityPage  .
 *   `component-info.yaml`   `spec.type`
 *   .  serviceEntityPage `spec.type`
 * service  , websiteEntityPage `spec.type`
 * website  .
**/
const cloudForetCoreServiceEntityPage = (
  <EntityLayout>
    <EntityLayout.Route path="/" title="Overview">
      {overviewContent}
    </EntityLayout.Route>

    <EntityLayout.Route path="/docs" title="Docs">
      {techdocsContent}
    </EntityLayout.Route>

    <EntityLayout.Route path="/kubernetes" title="Kubernetes">
      <EntityKubernetesContent refreshIntervalMs={30000} />
    </EntityLayout.Route>
  </EntityLayout>
);

/**
 *    componentPage  cloudForetCoreServiceEntityPage
 * .    `spec.type`     .
**/
const componentPage = (
  <EntitySwitch>
    <EntitySwitch.Case if={isComponentType('service')}>
      {serviceEntityPage}
    </EntitySwitch.Case>

    /*   */
    <EntitySwitch.Case if={isComponentType('coreservice')}>
      {cloudForetCoreServiceEntityPage}
    </EntitySwitch.Case>
    /*   */
```

```
      <EntitySwitch.Case if={isComponentType('website')}>
        {websiteEntityPage}
      </EntitySwitch.Case>

      <EntitySwitch.Case>{defaultEntityPage}</EntitySwitch.Case>
    </EntitySwitch>
  );
```

(2) 백엔드 코드 설정해주기

- 이제 backend 쪽 코드를 설정해줘야 합니다.
  이 부분은 공식 문서 그대로 진행해도 되니, 문서를 참고해주시길 바랍니다.
- 공식 문서

(3) Kubernetes 설정하기

- 이 부분에서는 Backstage Kubernetes plugin이 EKS cluster에 접근해 원하는 정보를 가져오기 위한 설정 작업을 진행합니다. 필요한 정보는 아래와 같고, `app-config.<>.yaml` 파일에 지정합니다.

```
kubernetes:
  serviceLocatorMethod:
    type: 'multiTenant' # multiTenant      .
  clusterLocatorMethods:
    - type: 'config'
      clusters:
          # url `kubectl cluster-info`  Kubernetes Control Plane .
        - url: https://kubernetes-control-plane-url.com
          # name   .
          name: spaceone-doodle-cluster-v1
          # aws, gke    ,     authProvider
          #      Kubernetes serviceAccount .
          authProvider: 'serviceAccount'
          skipTLSVerify: true
          skipMetricsLookup: false
          serviceAccountToken: ${K8S_SERVICEACCOUNT_TOKEN}
          caData: ${K8S_CA_DATA}
```

- 위에서는 두 개의 환경 변수를 사용하는데, 이 부분은 serviceAccount를 발급할 때 사용하는 토큰 및 CA 정보를 입력하면 됩니다. 기존에 사용되던 `backstage-secret` 에 아래와 같이 추가합니다.

```
# backstage-secret.yaml
apiVersion: v1
kind: Secret
#..
data:
  #..
  K8S_SERVICEACCOUNT_TOKEN: <base64-encoded-serviceAccountToken>
  K8S_CA_DATA: <base64-encoded-ca-data>
```

(4) metrics-server 설치하기

- Backstage는 Kubernetes에 배포된 pod의 상태를 불러오기 위해, 기본적으로 metrics-server가 cluster에 설치되어 있다고 가정합니다.
- 간단 설치로, 아래 명령어로 설치합니다.

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server
/releases/latest/download/components.yaml
```

(5) Kubernetes에 배포한 Backstage Component를 연동하기

- 이 부분은 Kubernetes에 배포한 컴포넌트를 Backstage에서 파악하기 위해 연동하는 과정입니다.
  Backstage가 사용하는 label을 추가해야 하며, cloudforet core 서비스의 경우, 두 부분에 추가해야 합니다.
  - 각 서비스의 `backstage/component-info.yaml`
  - 각 서비스의 helm chart
- identity 서비스를 기준으로 설명합니다.
- 먼저 `backstage/component-info.yaml` 입니다.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: cloudforet-identity
  namespace: cloudforet-core
  annotations:
    backstage.io/techdocs-ref: dir:.
    backstage.io/managed-by-origin-location: url:https://github.com
/sang-w0o/identity/blob/master/backstage/component-info.yaml
    #  backstage.io/kubernetes-*  3 .
    backstage.io/kubernetes-id: cloudforet-identity
    backstage.io/kubernetes-namespace: doodle-core
    backstage.io/kubernetes-label-selector: 'backstage.io
/kubernetes-id=cloudforet-identity'
spec:
  type: coreService
  lifecycle: experimental
  owner: whdalsrnt@mz.co.kr
```

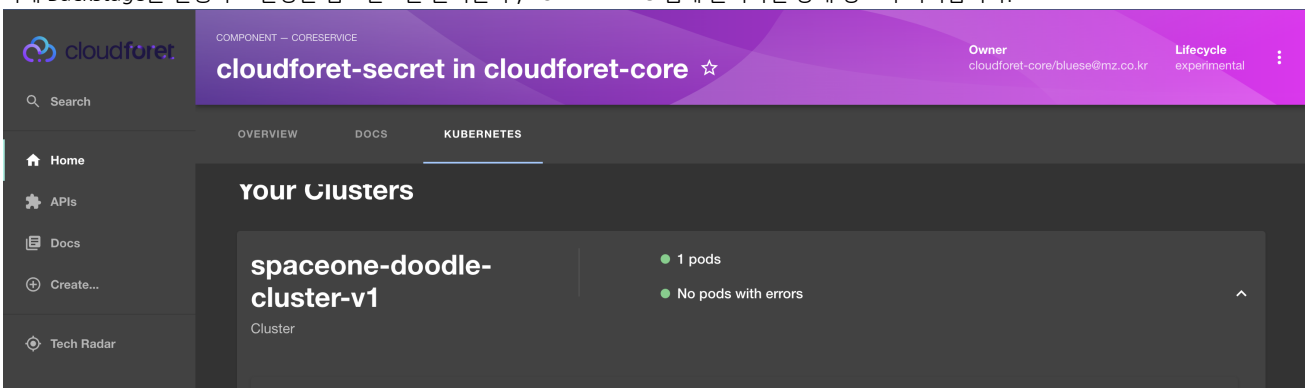
- yaml 파일을 보면 3개의 어노테이션이 추가되었으며, 각각에 대한 설명은 아래와 같습니다.
  - `backstage.io/kubernetes-id` : Backstage가 연결된 Kubernetes에서 컴포넌트를 찾기 위한 설정
  - `backstage.io/kubernetes-namespae`: 찾을 컴포넌트가 위치한 kubernetes namespace
  - `backstage.io/kubernetes-label-selector`: 컴포넌트에 설정한 label.

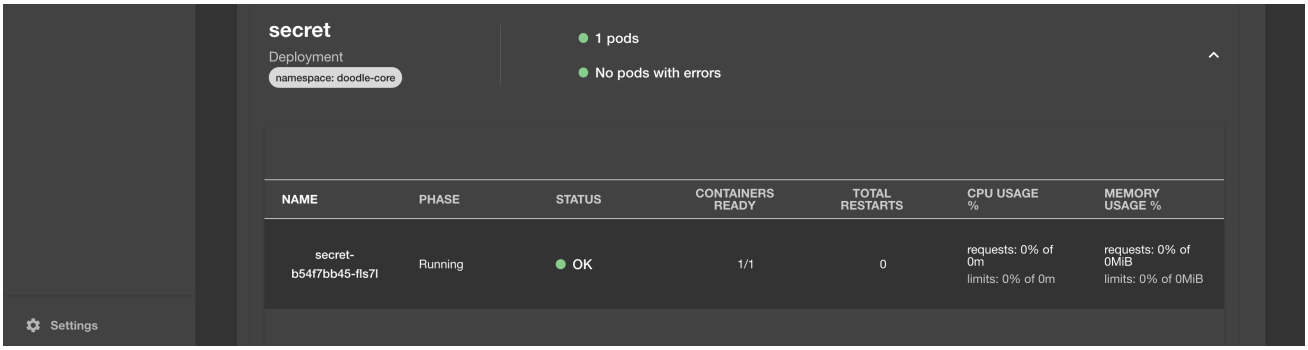
- 위에서 `backstage.io/kubernetes-label-selector` 를 달아주었고, 그 값으로 `'backstage.io/kubernetes-id=cloudforet-identity'` 를 지정해주었으니, 실제 Kubernetes object에도 이 label을 적용해야 합니다.

  Identity service를 기준으로, 이 설정을 `deploy/helm/templates/deployment-grpc.yaml` 에 지정합니다. 참고로, deployment 자체 및 `spec.template.label` 에도 지정해야 합니다!
  (deployment에만 설정하면 Backstage가 pod를 찾지 못합니다.)

```
# deploy/helm/templates/deployment-grpc.yaml
{{- if .Values.enabled }}
{{- if .Values.grpc }}

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    #  label
    backstage.io/kubernetes-id: cloudforet-identity
{{ include "spaceone.labels" . | indent 4 }}
    spaceone.service: {{ .Values.name }}
  name: {{ .Values.name }}
  namespace: {{ .Values.global.namespace | default .Release.
Namespace }}
spec:
  replicas: {{ .Values.replicas }}
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      spaceone.service: {{ .Values.name }}
  template:
    metadata:
      annotations:
        spaceone.deployment.tool: helm
        spaceone.database-flag: {{ print .Values.database |
sha256sum }}
        spaceone.shared-flag: {{ print .Values.global.shared |
sha256sum }}
        spaceone.application-flag: {{ print .Values.
application_grpc | sha256sum }}
      labels:
        #  label
        backstage.io/kubernetes-id: cloudforet-identity
{{ include "spaceone.labels" . | indent 8 }}
        spaceone.service: {{ .Values.name }}
    spec:
      #..
```

- 이제 Backstage를 실행하고 설정한 컴포넌트를 클릭한 후, KUBERNETES 탭에 들어가면 상태 정보가 나타납니다.





TechDocs를 S3에 저장해 사용하기

- Backstage는 markdown 파일을 렌더링하기 위해 `mkdocs` 라는 라이브러리를 사용하며, markdown 파일을 HTML로 빌드하기 위한 총 세 가지 방법을 제공합니다.
  - Docker container를 띄워서 빌드
  - 로컬에서 빌드
  - CI/CD 파이프라인에서 빌드 후 Cloud storage에 저장: 이 방식이 권장되는 방식입니다.
- 여기서는 "CI/CD 파이프라인에서 빌드 후 Cloud storage에 저장" 하는 방식에 대해 살펴볼 것이며, cloud storage로는 Amazon S3를 사용할 것입니다.

1. IAM 사용자 발급하기

- S3 bucket을 생성하고, 아래의 inline policy를 가지는 IAM user를 생성합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TechDocsWithMigration",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObjectVersion",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::bucket_",
                "arn:aws:s3:::bucket_/*"
            ]
        }
    ]
}
```

- 해당 IAM user의 access key ID, secret access key를 사용해야 합니다.

2. 각 component 파일에 Github Actions Workflow 지정하기

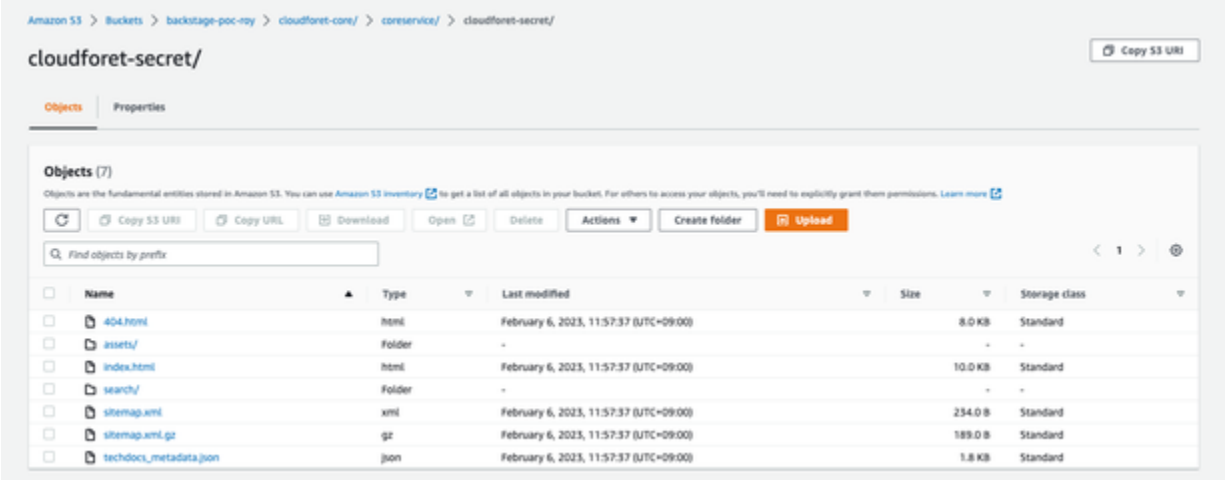
- 공식 문서

- 문서는 각 component repository에 markdown으로 저장되기에, 문서에 변경 사항이 생겼을 때 다시 파일을 생성하는 주체는 해당 component repository여야 합니다.
- 아래와 같이 `.github/workflows/publish-techdocs.yml` 파일을 작성합니다.
(`console-api-v2` 서비스의 예시 기준입니다.)

```
name: Publish TechDocs to S3

on:
  push:
    branches:
      - master
  workflow_dispatch:

jobs:
  publish-techdocs-s3:
    runs-on: ubuntu-latest
    env:
      AWS_ACCOUNT_ID: ${{ secrets.AWS_ACCOUNT_ID }}
      AWS_REGION: ${{ secrets.AWS_REGION }}
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      AWS_S3_BUCKET_NAME: ${{ secrets.AWS_S3_BUCKET_NAME }}
      ENTITY_NAMESPACE: 'cloudforet-core'
      ENTITY_KIND: 'coreService'
      ENTITY_NAME: 'cloudforet-console-api-v2'

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - uses: actions/setup-node@v3
      - uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: setup techdocs-cli
        run: sudo npm install -g @techdocs/cli

      - name: install mkdocs and mkdocs plugins
        run: python -m pip install mkdocs-techdocs-core==1.*

      - name: Generate docs
        run: techdocs-cli generate --source-dir ./backstage --no-
docker --verbose

      - name: Publish docs
        run: techdocs-cli publish --publisher-type awsS3 --storage-
name $AWS_S3_BUCKET_NAME --entity $ENTITY_NAMESPACE/$ENTITY_KIND
/$ENTITY_NAME &&
```

```
                techdocs-cli publish --publisher-type awsS3 --storage-
name $AWS_S3_BUCKET_NAME --entity $ENTITY_NAMESPACE/component
/$ENTITY_NAME
```

- 위 파일에서는 총 5개의 github repository secret을 환경변수로 주입받아 사용합니다.
  - `AWS_ACCOUNT_ID`: AWS 계정의 숫자로 구성된 ID 입니다. (ex. `277565498729`)
  - `AWS_REGION`: region 코드로, 서울의 경우 `ap-northeast-2` 입니다.
  - `AWS_ACCESS_KEY_ID` : 1단계에서 발급받은 IAM user의 access key ID 입니다.
  - `AWS_SECRET_ACCESS_KEY` : 1단계에서 발급받은 IAM user의 secret access key 입니다.
  - `AWS_S3_BUCKET_NAME` : 1단계에서 생성한 문서를 저장하기 위한 S3 bucket의 이름입니다. (arn이 아닙니다)
- 그 외 `ENTITY_NAMESPACE`, `ENTITY_KIND`, `ENTITY_NAME` 은 컴포넌트에 알맞게 설정해줘야 합니다.
- 성공적으로 workflow가 실행되면, S3 bucket에 아래처럼 저장됩니다.
  아래 사진은 `ENTITY_NAMESPACE`가 `cloudforet-core`, `ENTITY_KIND` 가 `coreService`, 그리고 `ENTITY_NAME` 이 `cloudforet-secret` 인 workflow의 결과 예시입니다.



3. Backstage 설정하기

- 공식 문서
- 이제 Backstage가 S3에서 문서를 읽어와 `docs` 페이지에 렌더링하도록 해야 합니다.
  아래처럼 `app-config.yaml` 을 수정합니다.

```
techdocs:
  #  component docs S3   external .
  #   , 'local' .
  builder: 'external'
  generator:
    runIn: 'local'
  publisher:
    type: 'awsS3'
    awsS3:
      bucketName: ${AWS_S3_BUCKET_NAME}
      accountId: ${AWS_ACCOUNT_ID}
      region: ${AWS_REGION}
aws:
  accounts:
```

```
    - accountId: ${AWS_ACCOUNT_ID}
      accessKeyId: ${AWS_ACCESS_KEY_ID}
      secretAccessKey: ${AWS_SECRET_ACCESS_KEY}
```

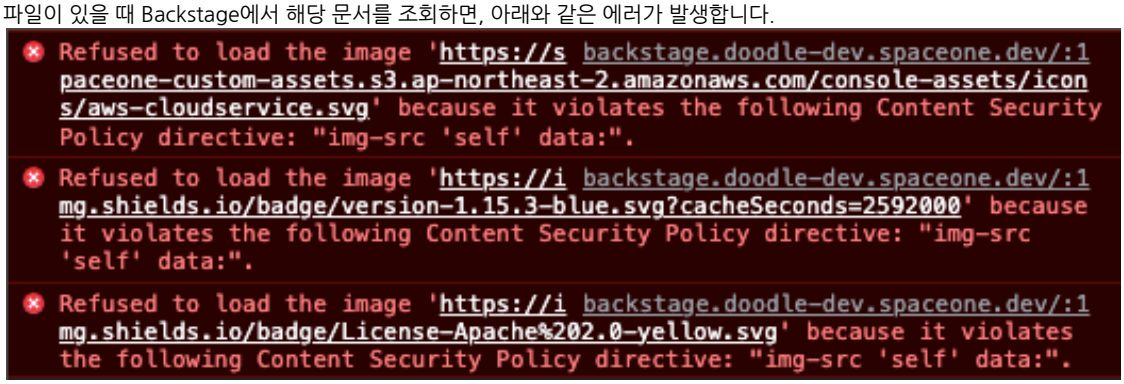
- 위에서 사용하는 환경 변수들은 모두 base64 encoding해 참조하는 Kubernetes secret에 저장해야 합니다.

4. (optional) markdown 파일에서 외부 이미지를 불러와 사용하는 경우

- plugin-aws-cloud-service-inven-collector 의 경우, README.md에서 아래와 같이 외부 이미지를 불러와 사용합니다.

```
<h1 align="center">AWS Cloud Service Collector</h1>

<br/>
<div align="center" style="display:flex;">
  <img width="245" src="https://spaceone-custom-assets.s3.ap-
northeast-2.amazonaws.com/console-assets/icons/aws-cloudservice.svg"
>
  <p>
    <br>
    <img alt="Version"  src="https://img.shields.io/badge/version-
1.15.3-blue.svg?cacheSeconds=2592000"  />
    <a href="https://www.apache.org/licenses/LICENSE-2.0"  target="
_blank"><img alt="License: Apache 2.0"  src="https://img.shields.io
/badge/License-Apache 2.0-yellow.svg" /></a>
  </p>
</div>
```

- 위와 같은 파일이 있을 때 Backstage에서 해당 문서를 조회하면, 아래와 같은 에러가 발생합니다.



- 이 에러가 발생하는 이유는 기본적으로 Backstage의 Content Security Policy(CSP)가 `self` , 즉 자기 자신에서만 데이터를 받아오도록 설정되어 있기 때문입니다.
- 이를 해결하기 위해 CSP의 `img-src`에 url을 하나씩 넣어줄 수도 있지만, 앞으로 새로운 문서에서 다양한 url을 참조하게 될 때 일일이 추가해야 하는 번거로움이 있으므로 `app-config.production.yaml` 을 아래와 같이 수정합니다.

```
backend:
  #..
  csp:
    connect-src: ["'self'", 'http:', 'https:']
    img-src: ["'self'", 'data:', 'https:']
```

- 이제 정상적으로 `https` protocol을 사용하는 url로부터 이미지를 불러올 수 있습니다.

Backstage에 서비스의 Swagger 적용하기

1. Backstage frontend 설정하기

- app 에 @backstage/plugin-api-docs 를 설치해 사용합니다.
- 설치 방법은 아래와 같습니다.
  (`npx @backstage/create-app`으로 생성한 프로젝트에는 기본 포함되어 있습니다.)

```
yarn add --cwd packages/app @backstage/plugin-api-docs
```

- 다음으로 `Component Kubernetes - (1)` 와 마찬가지로 api를 위한 냉 `packages/app/src/components/catalog/EntityPage.tsx` 에 아래 코드를 추가합니다.

```
const cloudForetCoreServiceEntityPage = (
  <EntityLayout>
    <EntityLayout.Route path="/" title="Overview">
      {overviewContent}
    </EntityLayout.Route>

    <EntityLayout.Route path="/docs" title="Docs">
      {techdocsContent}
    </EntityLayout.Route>

    <EntityLayout.Route path="/kubernetes" title="Kubernetes">
      <EntityKubernetesContent refreshIntervalMs={30000} />
    </EntityLayout.Route>

    <EntityLayout.Route path="/api" title="API">
      <Grid container spacing={3} alignItems="stretch">
        <Grid item md={6}>
          <EntityProvidedApisCard />
        </Grid>
        <Grid item md={6}>
          <EntityConsumedApisCard />
        </Grid>
      </Grid>
    </EntityLayout.Route>
  </EntityLayout>
);
```

2. `component-info.yaml` 에 API 설정 해주기

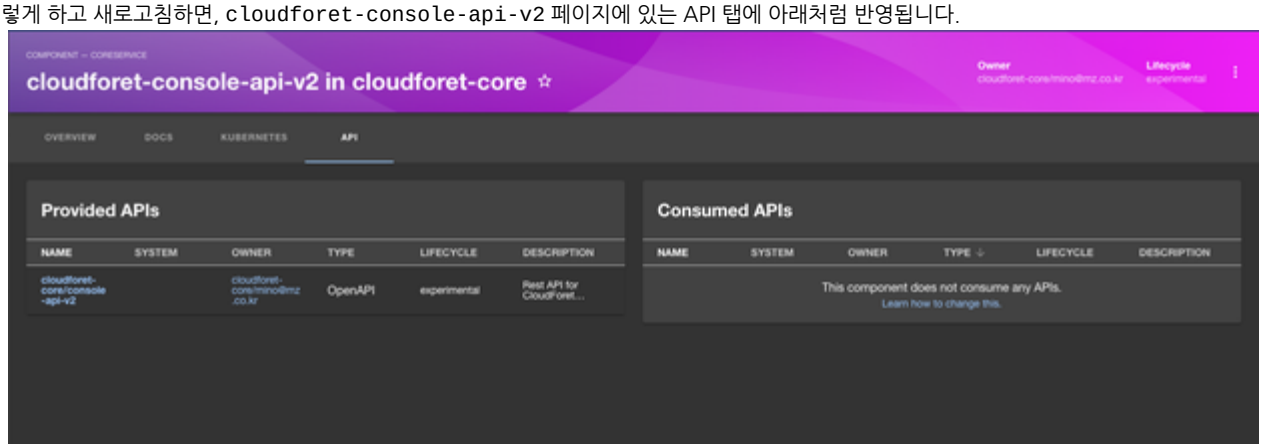
- 아래와 같이 openapi spec을 담은 JSON 파일이 저장되어 있다고 가정합니다.

```
|-- backstage
      |-- docs
            |-- index.md
      |-- api
            |-- console-api-v2.json
      |-- component-info.yaml
      |-- mkdocs.yml
```

- 기존에 `kind: Component` 인 yaml 파일을 만들어 Backstage가 Component로 인식하도록 했던 것처럼, 이번에는 `kind: API` 인 yaml 파일을 생성해 Backstage가 API로 인식할 수 있도록 해야 합니다.
- 아래와 같이 추가합니다.

```yaml
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: cloudforet-console-api-v2
  #..
spec:
  #..
  # spec.providesApi  cloudforet-console-api-v2
  # console-api-v2 API  Backstage .
  providesApi: [console-api-v2]
---
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: console-api-v2
  #..
spec:
  type: openapi
  lifecycle: experimental
  owner: mino@mz.co.kr
  # definition $text openapi spec  JSON  YAML
  #   .
  definition:
    $text: ./api/console-api-v2.json
```

- 이렇게 하고 새로고침하면, `cloudforet-console-api-v2` 페이지에 있는 API 탭에 아래처럼 반영됩니다.



- `Provided APIs` 에서 클릭하거나, 좌측의 `APIs` 탭을 통해 상세 정보를 확인할 수 있습니다.