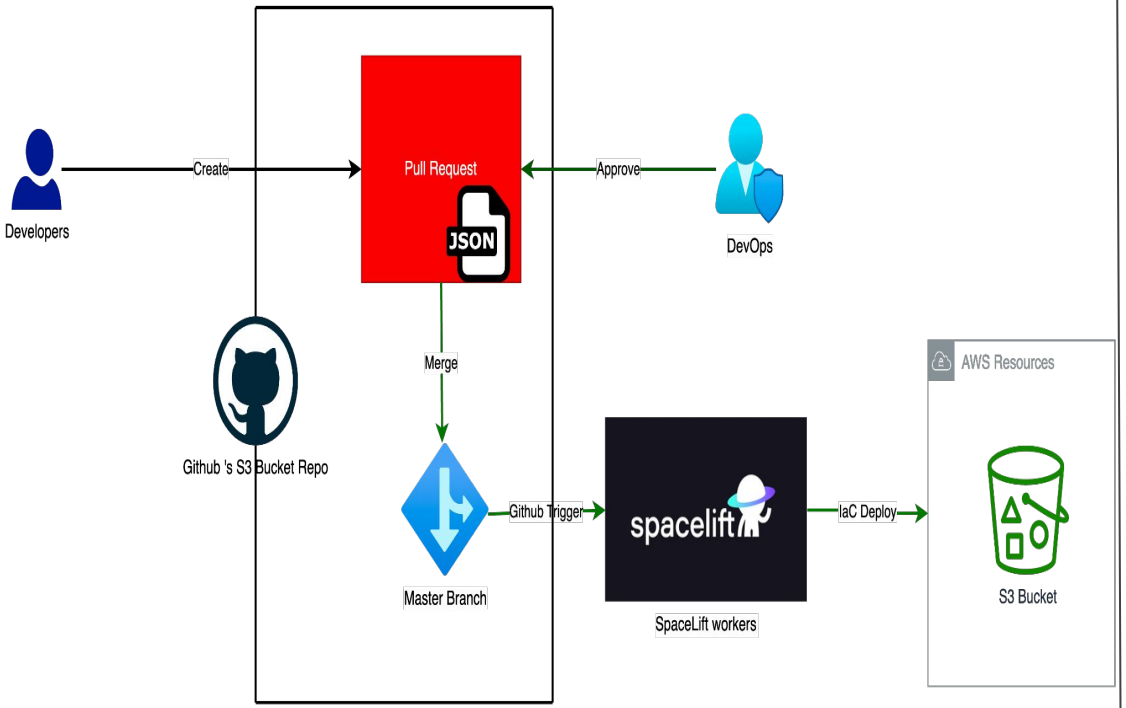
IAG Talents Day

Presenter : Shan Nam

Agenda

- Introduce Myself
- Self-service
- Backstage core features:
  - Service Catalog
  - Software Template
- Backstage in real problem :
  - Scenario 1 : Onboard 1st party-Service
  - Scenario 2 : Provision AWS resources

Self-service

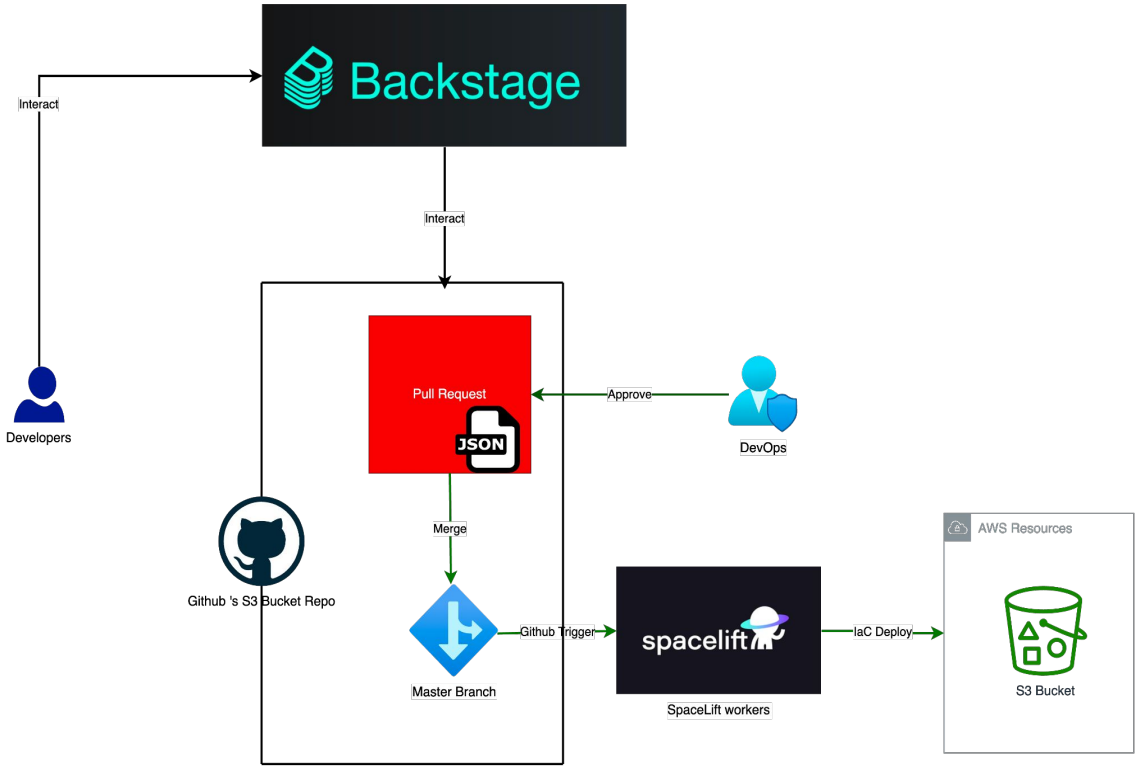


Disadvantages:

- Not-so-Friendly interface
  -> Leads to problem Is : Most of PR are created by DevOps team and also reviewed, merged by DevOps team

- The only Mechanic to manage Metadata of resources is using attached Tags :
  -> Hard to manage the Relationship between Application - AWS resource

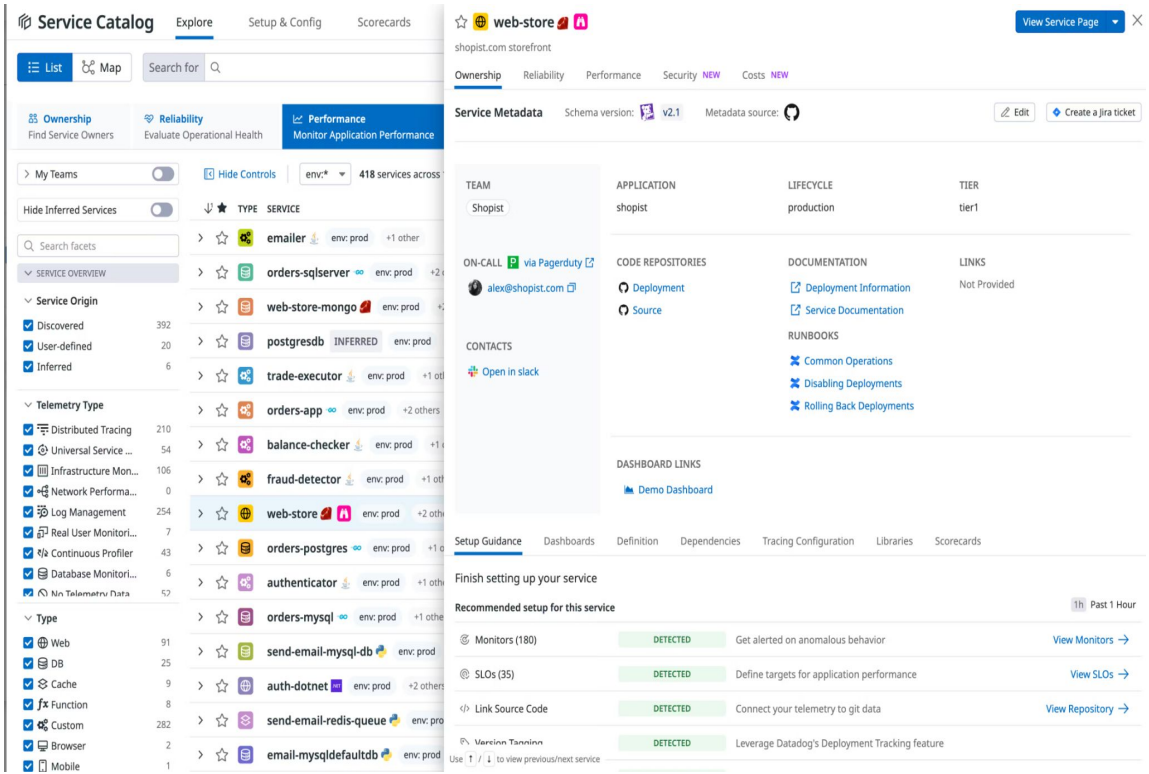
Self-service



Backstage comes to rescue:

- Provides Unified Service Catalog Dashboard which tracks down all created Applications, Resources, Metadata of Them, Visualize dependencies between different components/resources.
- Provide Powerful “Template” feature
- It is a sandbox which give us capability to customize every part of it

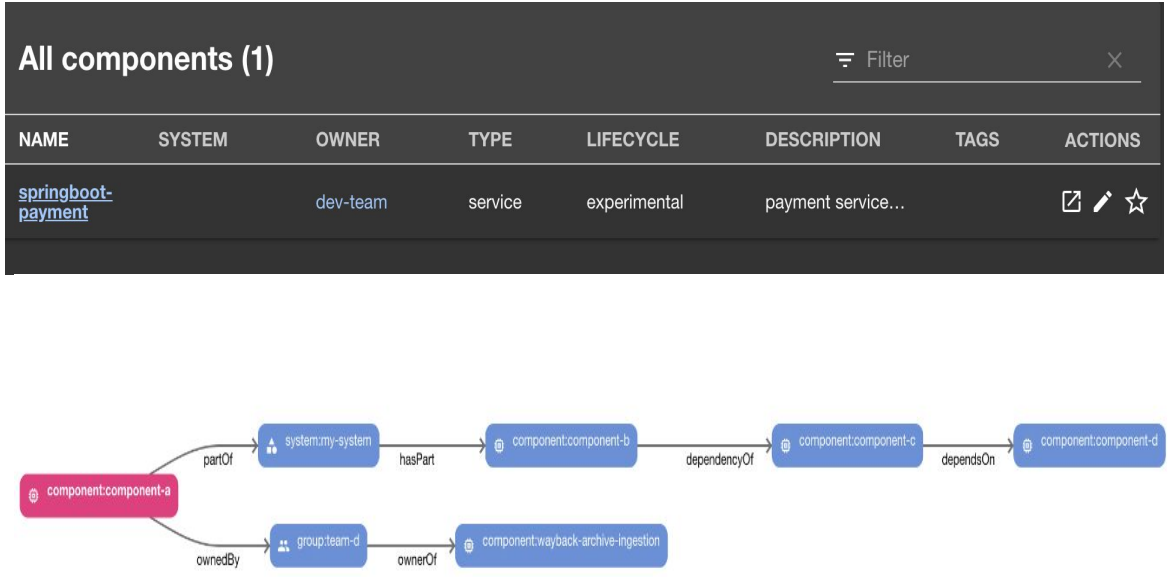
Service Catalog : From Datadog



Powerful, Provides useful information :

- Service Ownership
- Dependencies between different Applications : Using Telemetry Data
- Useful metrics

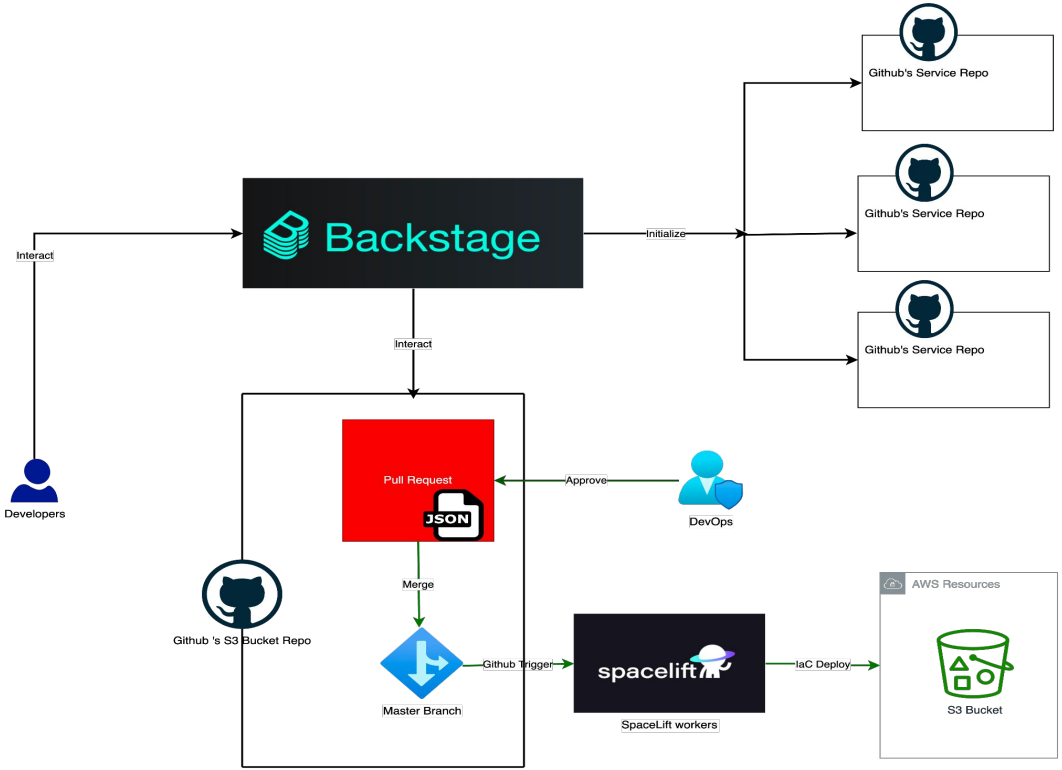
Service Catalog : From Backstage



Provide useful information to Dev team :

- OwnerShip, Pipeline's status, OpenAPI Visualization, Docs.
- More meaningful relationships in Dependencies graph : App-App, App-resources, App-user/group
- Sandbox-like capacity: Can be customized, even import Metrics dashboard from, for example, Datadog

Software template

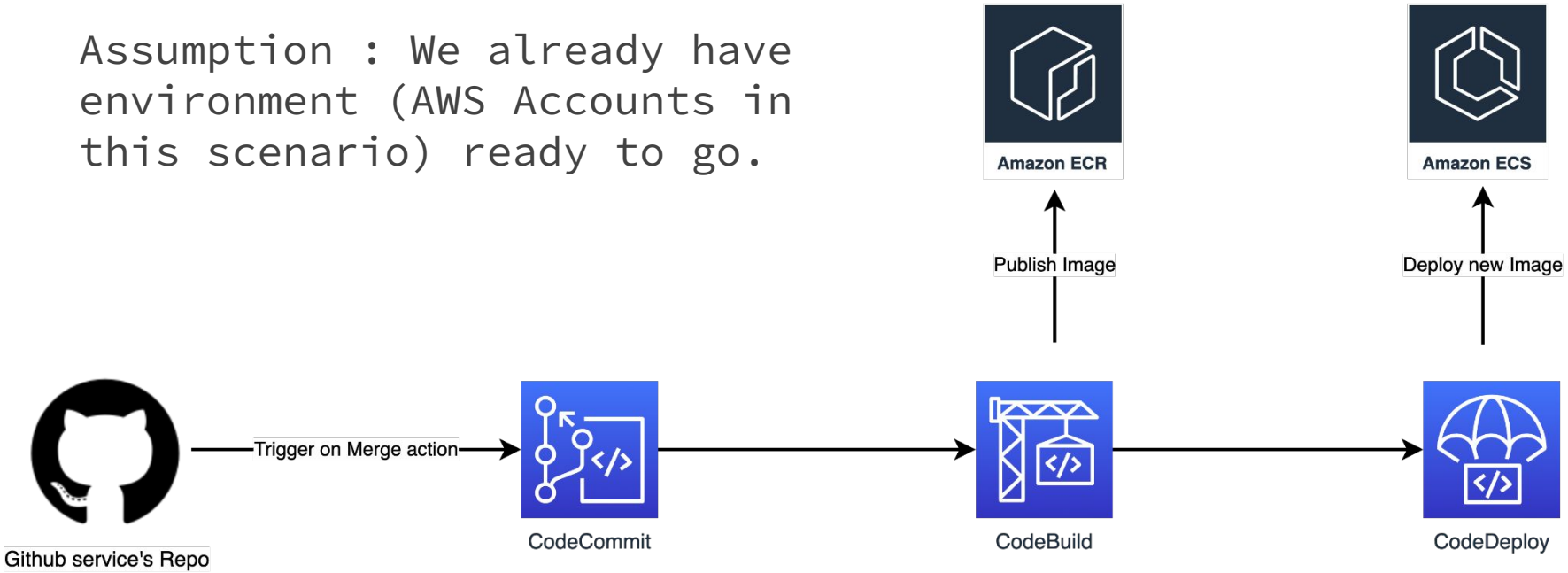


Not be limited to “template engine”

- Provide useful actions :
  - Create new Github Repository, create PR…
  - Ability to create our own customized action

Scenario : Onboard 1st party-Service

Assumption : We already have environment (AWS Accounts in this scenario) ready to go.

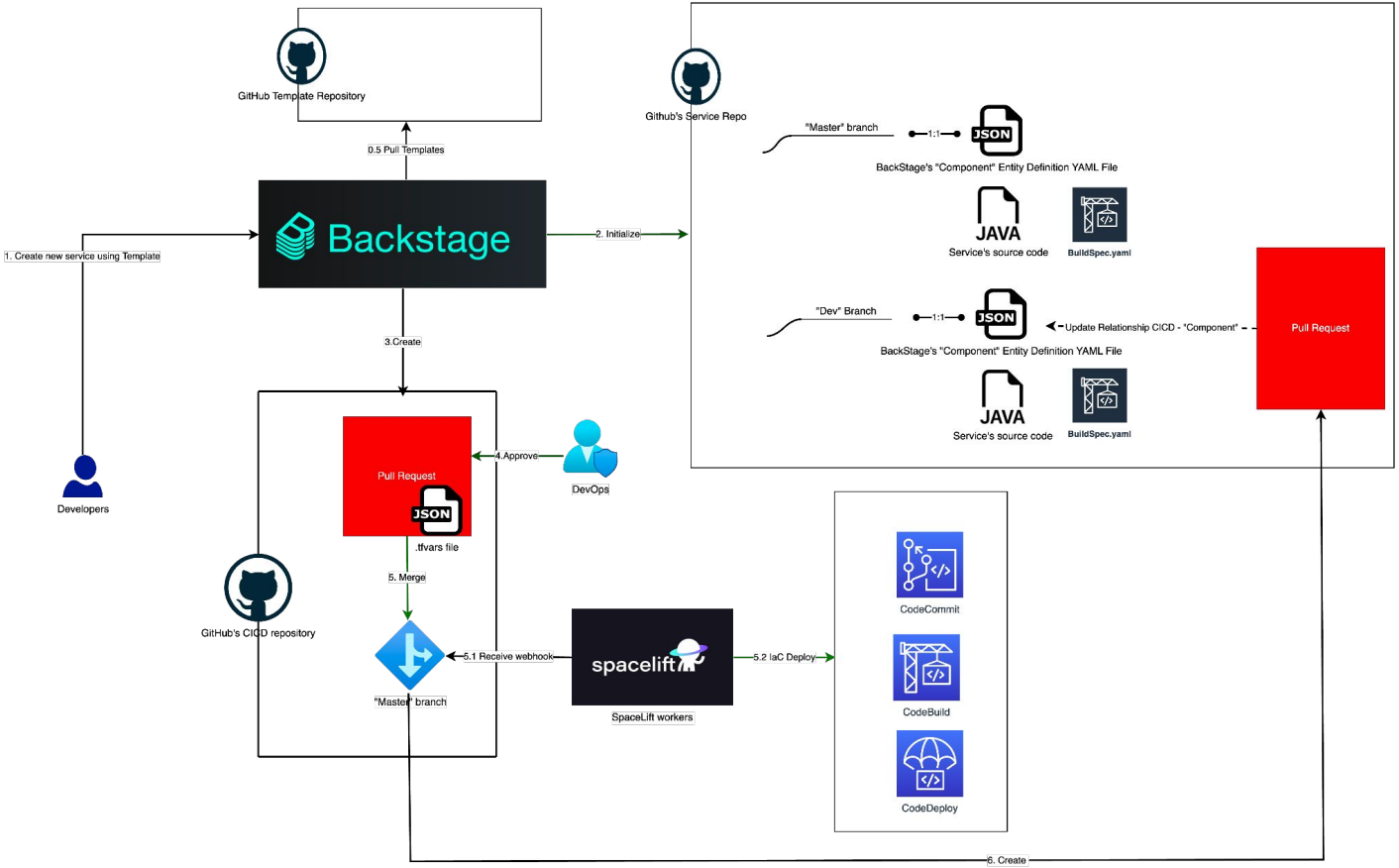


Scenario 1 : Onboard 1st party-Service

Requirements:

- Single Entry-point of onboard new Service : Backstage-Template
- The Result we would like to get :
  - New Repository for service : Master, Dev branches
  - CICD pipeline is already hooked up to Service’s Repo
  - “Service Catalog” is able to discover new onboarded Service
  - Visualization of status of Pipelines belong to each individual service in service catalog

Workflow :


GitHub Template Repository
0.5 Pull Templates
Backstage
1. Create new service using Template
2. Initialize
3.Create
Github's Service Repo
"Master" branch
1:1
JSON
BackStage's "Component" Entity Definition YAML File
JAVA
Service's source code
BuildSpec.yaml
"Dev" Branch
1:1
JSON
Update Relationship CICD - "Component"
Pull Request
BackStage's "Component" Entity Definition YAML File
JAVA
Service's source code
BuildSpec.yaml
Developers
Pull Request
JSON
4.Approve
DevOps
.tfvars file
5. Merge
GitHub's CICD repository
"Master" branch
5.1 Receive webhook
spacelift
SpaceLift workers
5.2 IaC Deploy
CodeCommit
CodeBuild
CodeDeploy
6. Create

Github Integration

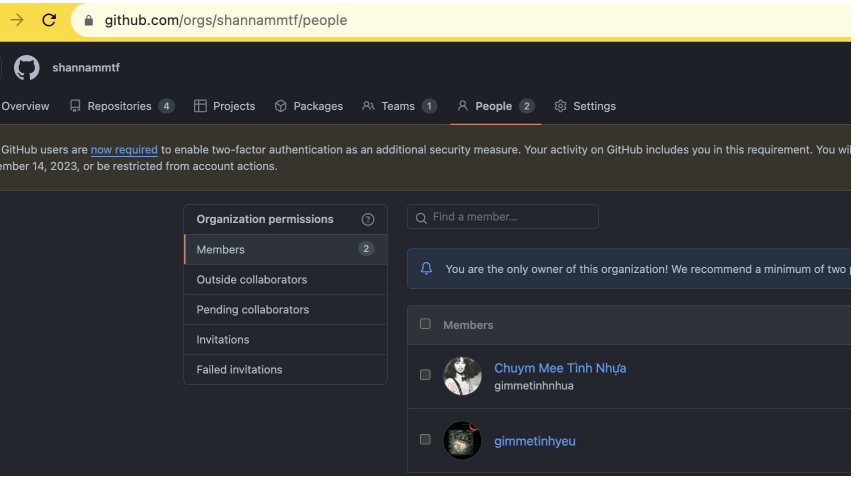
Provides :

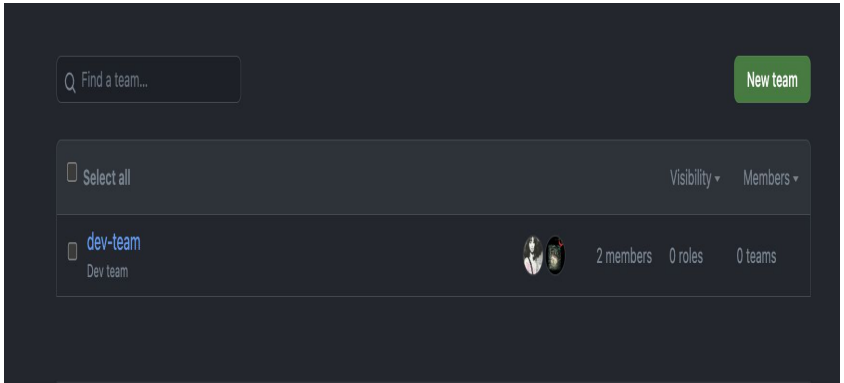
Ability to import Members/Groups In Organization to Backstage

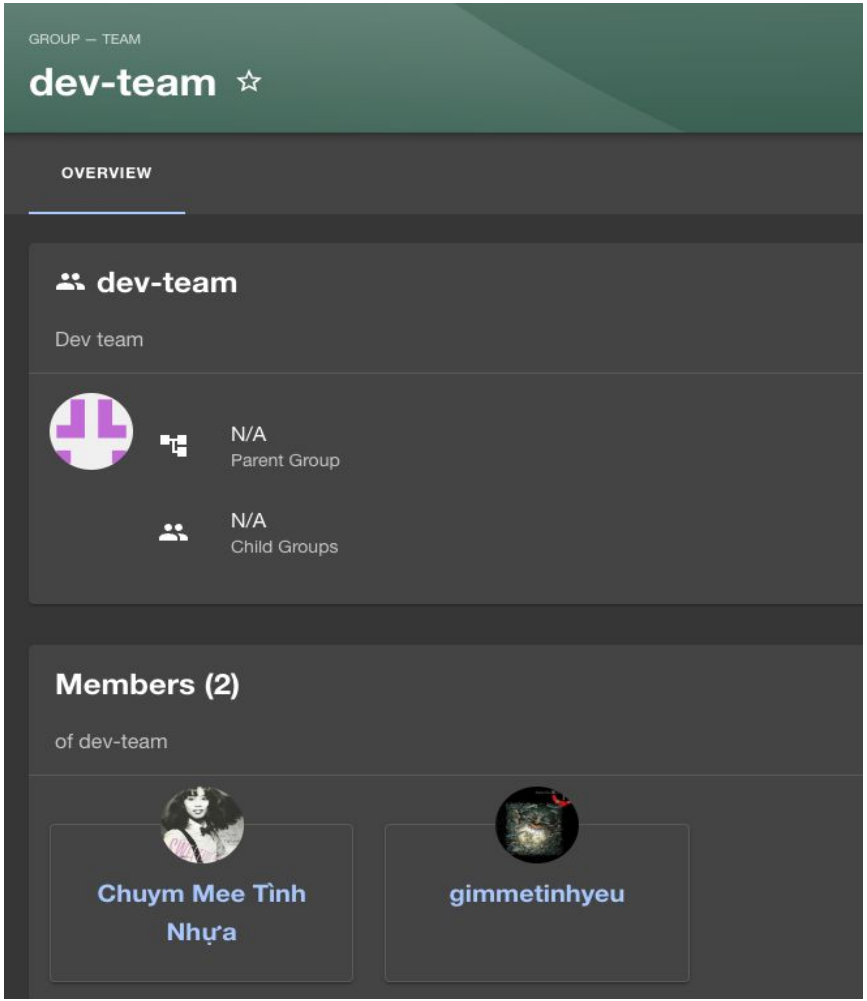
Automatically Scan All Repositories (or Selected one base on defined filters) and import Backstage Entity’s Definition file in each Repo into our Backstage backend

Act as Identity Provider, Authorization for actions made from “Software Templates” feature

Github Integration







Step 0.5 A Dedicated Repository contains different kinds of templates

---

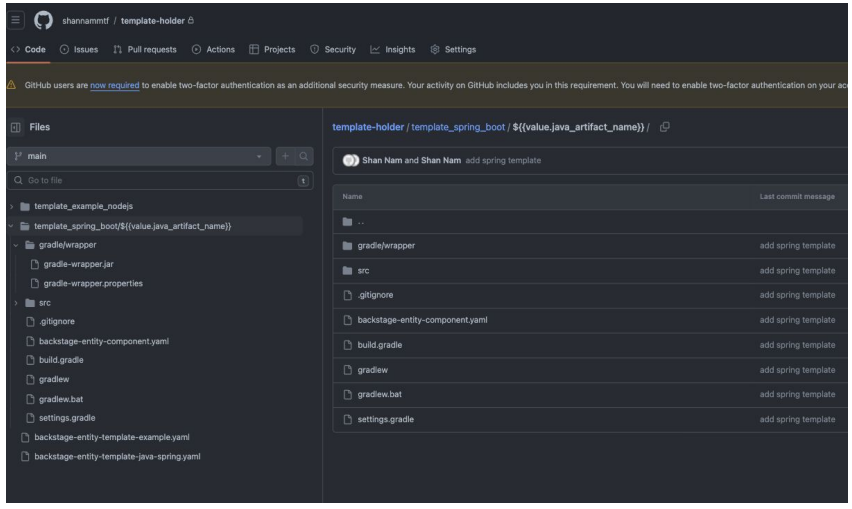
Reason : Version control

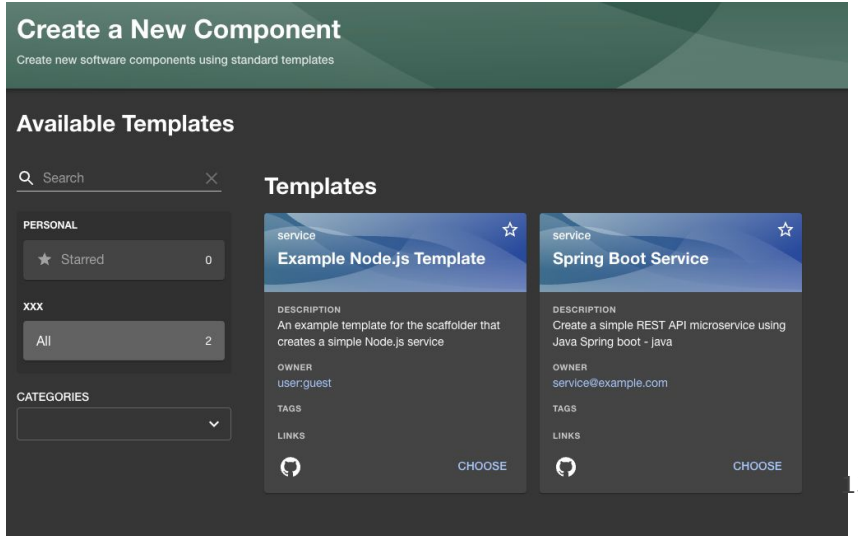
template-holder / backstage-entity-template-java-spring.yaml

Shan Nam and Shan Nam add pr action

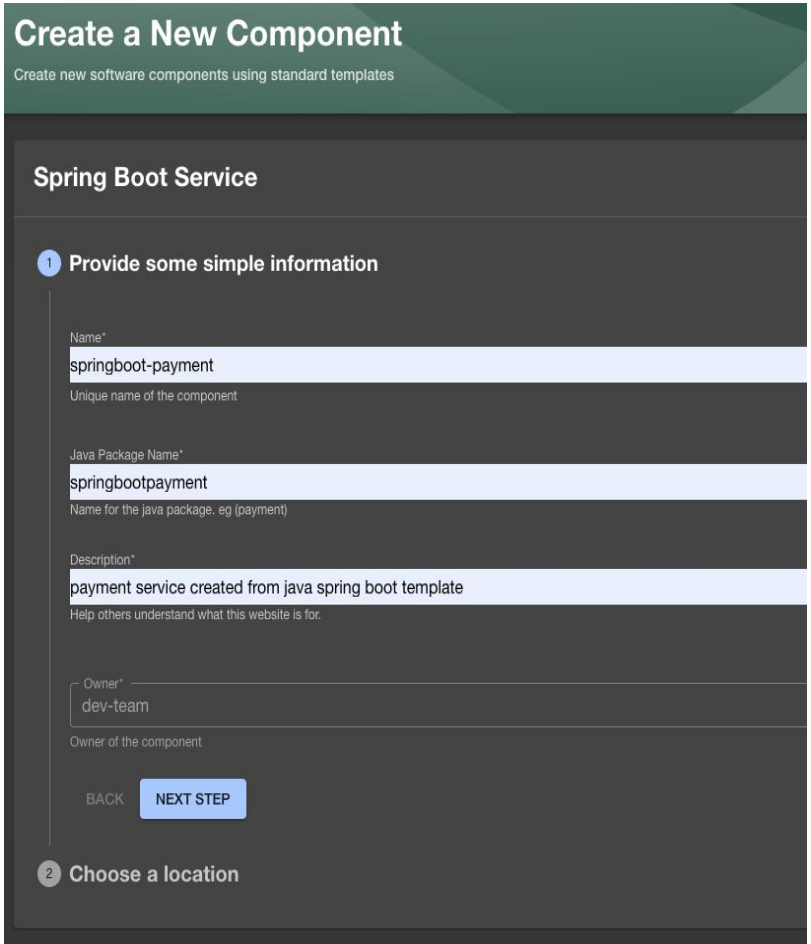
Code | Blame | 90 lines (85 loc) · 2.52 KB

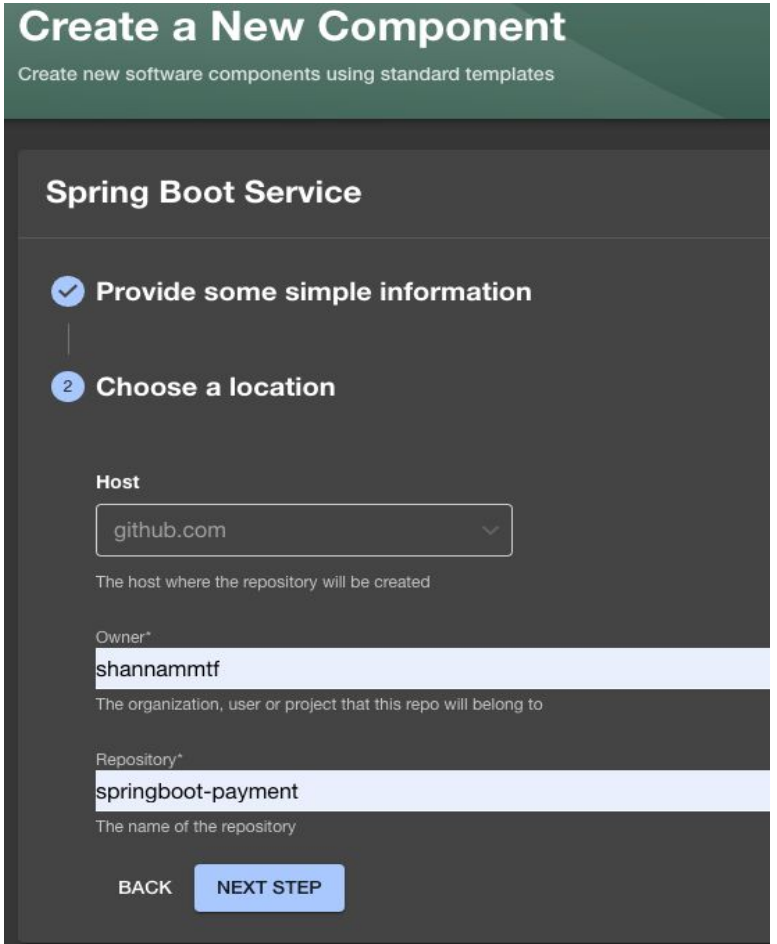
```yaml
apiVersion: scaffolder.backstage.io/v1beta3
kind: Template
metadata:
  name: springboot-template
  title: Spring Boot  Service
  description: Create a simple REST API microservice using Java Spring boot
    - java
spec:
  owner: service@example.com
  type: service

  parameters:
    - title: Provide some simple information
      required:
        - component_id
        - owner
        - java_package_name
        - description
      properties:
        component_id:
          title: Name
          type: string
          description: Unique name of the component
          ui:field: EntityNamePicker
        java_package_name:
          title: Java Package Name
          type: string
          description: Name for the java package. eg (payment)
        description:
          title: Description
          type: string
          description: Help others understand what this website is for.
        owner:
          title: Owner
          type: string
          description: Owner of the component
          ui:field: OwnerPicker
          ui:options:
            allowedKinds:
              - Group
```



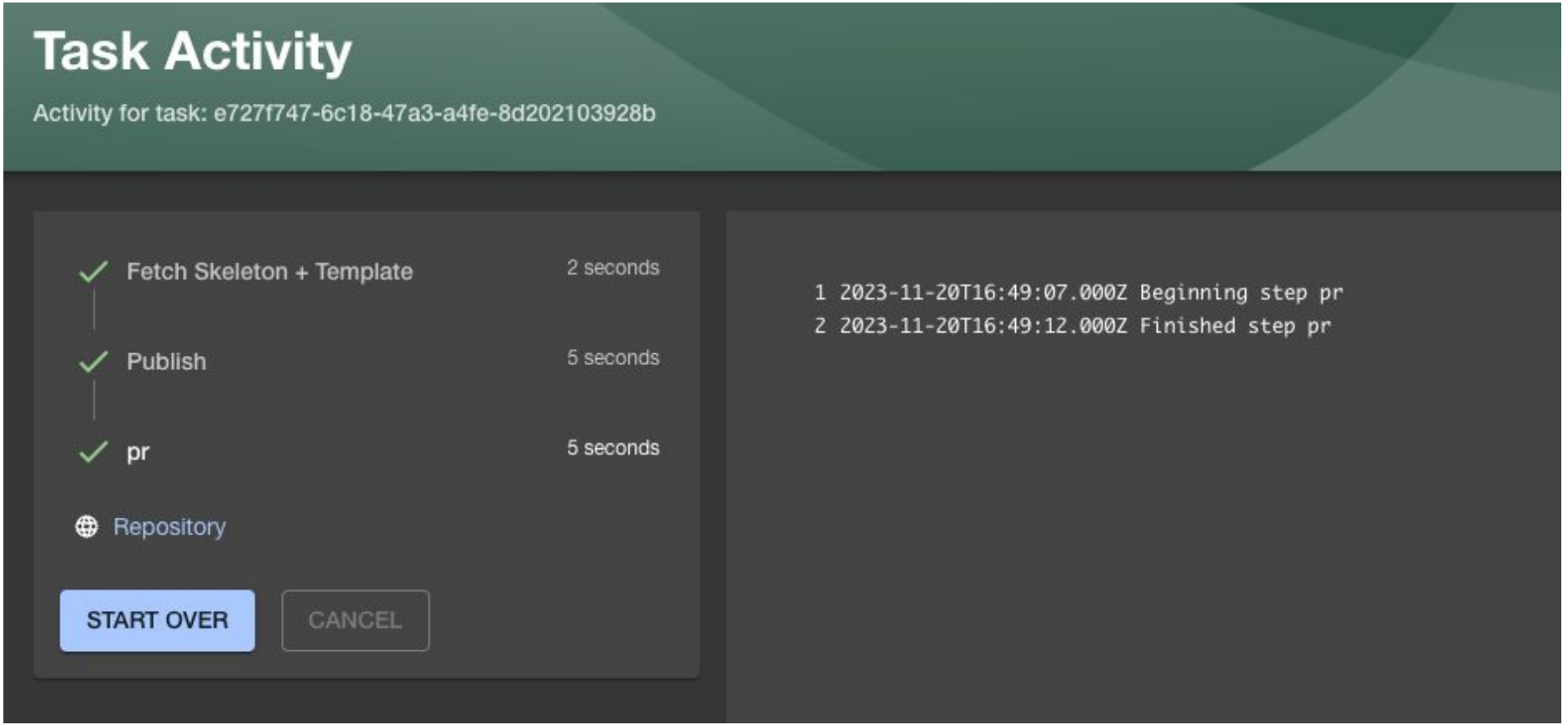


Step 1: Dev creates new Service base on predefined Template



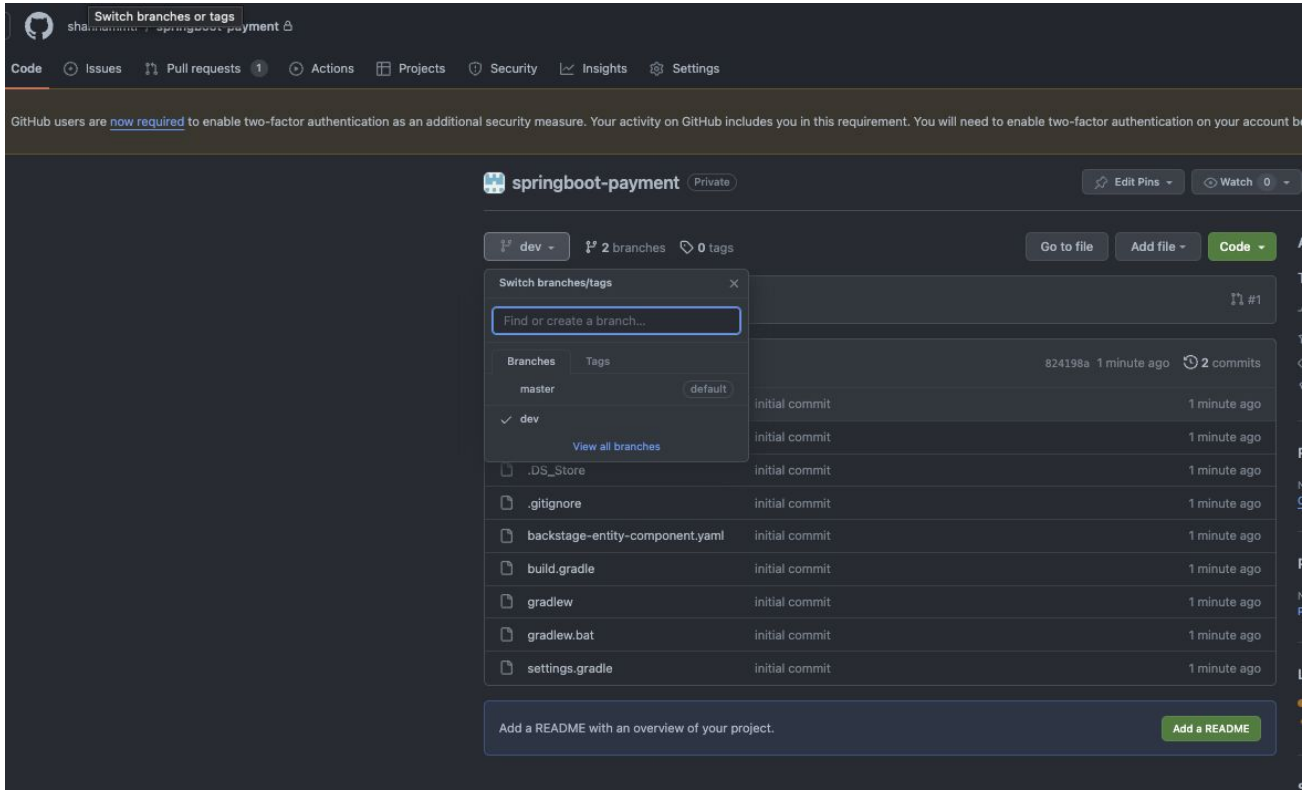


UI showed to the Dev after he hit “create” button



Result of step 2

Newly Created Service's Repository with its 2 branches : "master", "Dev"



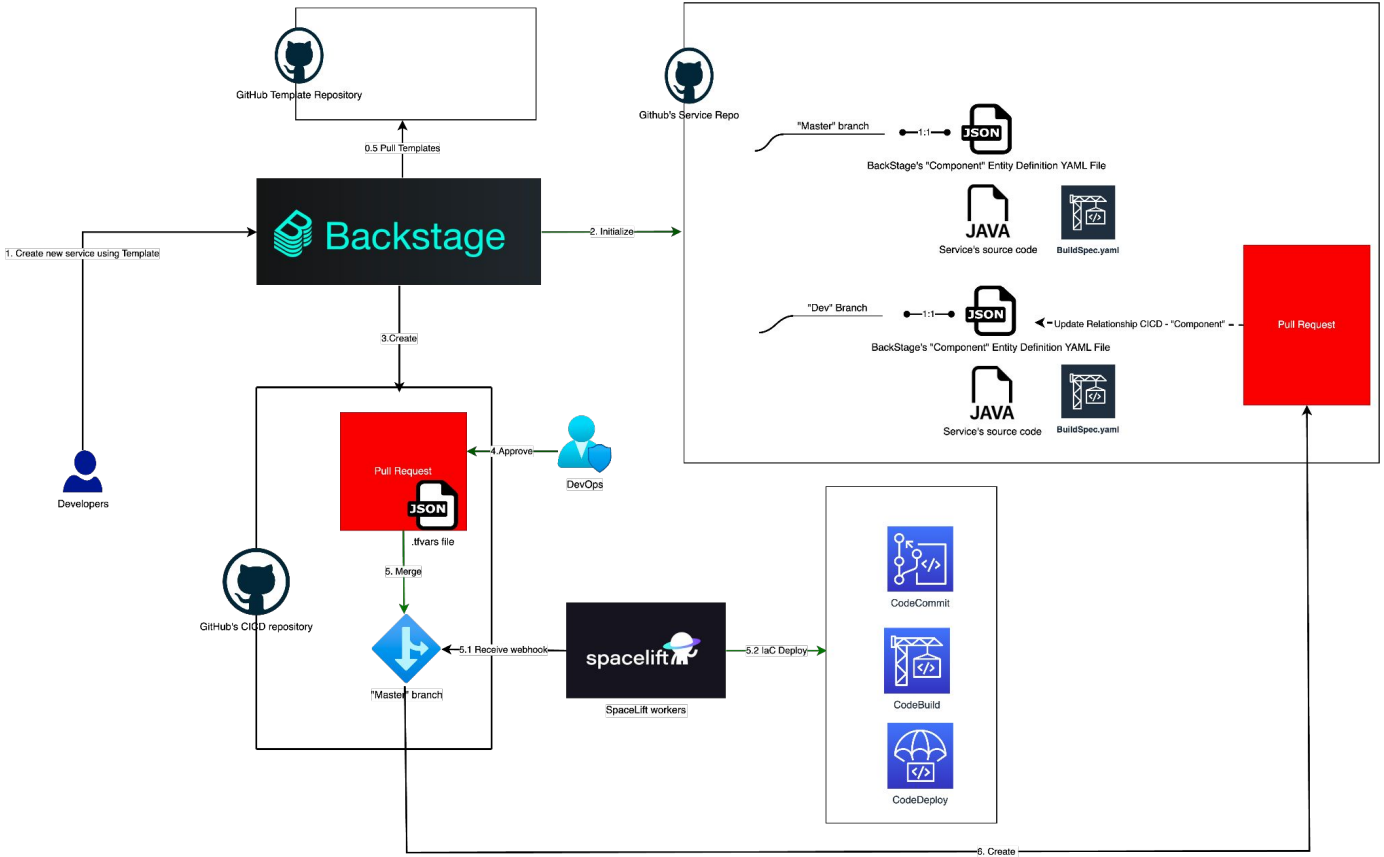
Step 2 under the hood

Under the hood, actions taken in order:

- Fetch template, create new source code for service based on param
- Create new Repo with single “master” branch
- Create a PR from “dev” branch to “master” branch which results in creating new “dev” branch

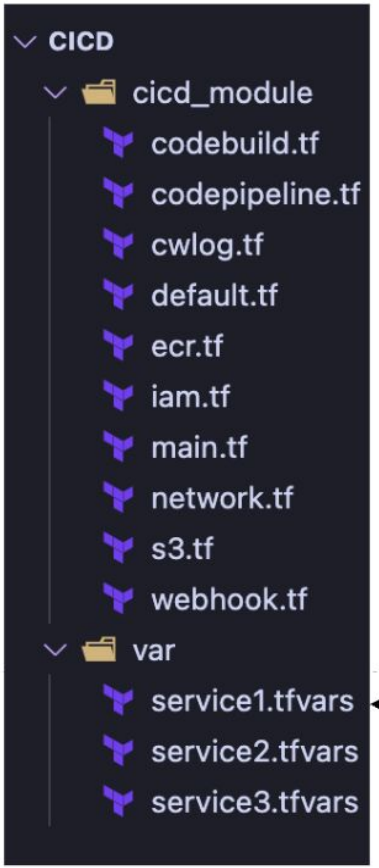
```yaml
steps:
  - id: template
    name: Fetch Skeleton + Template
    action: fetch:template
    input:
      url: https://github.com/shannammtf/template-holder/tree/dev/template_spring_boot/
      values:
        component_id: ${{ parameters.component_id }}
        description: ${{ parameters.description }}
        java_artifact_name: ${{ parameters.java_package_name }}
        owner: ${{ parameters.owner }}
        destination: ${{ parameters.repoUrl | parseRepoUrl }}


  - id: publish
    name: Publish
    action: publish:github
    input:
      allowedHosts: ["github.com"]
      description: This is ${{ parameters.component_id }}
      repoUrl: ${{ parameters.repoUrl }}

  - id: pr_create_branch
    name: pr
    action: publish:github:pull-request
    input:
      repoUrl: ${{ parameters.repoUrl }}
      branchName : dev
      targetBranchName : master
      title: to create new branch
      description : to create new branch
```

Workflow:


GitHub Template Repository
0.5 Pull Templates
Backstage
1. Create new service using Template
2. Initialize
3.Create
Github's Service Repo
"Master" branch
1:1
JSON
BackStage's "Component" Entity Definition YAML File
JAVA
Service's source code
BuildSpec.yaml
"Dev" Branch
1:1
JSON
Update Relationship CICD - "Component"
BackStage's "Component" Entity Definition YAML File
JAVA
Service's source code
BuildSpec.yaml
Pull Request
Developers
Pull Request
JSON
.tfvars file
4.Approve
DevOps
5. Merge
GitHub's CICD repository
"Master" branch
5.1 Receive webhook
spacelift
SpaceLift workers
5.2 IaC Deploy
CodeCommit
CodeBuild
CodeDeploy
6. Create

Idea for step 3 :

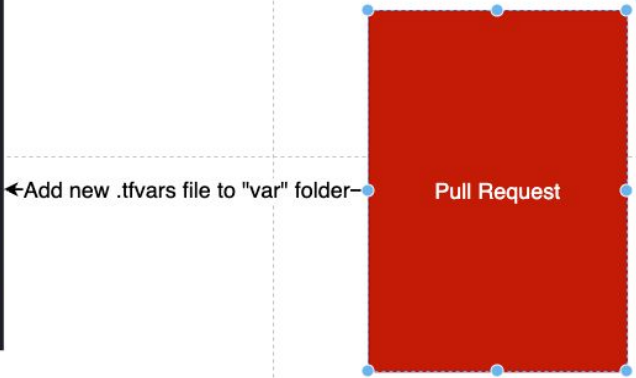
Dedicated Repo for CICD



Actions taken by Backstage:

1. Backstage fetches Template of CICD Config file and enriches it

2. Backstage Publishes a PR contains a new CICD config file to CICD Repo.



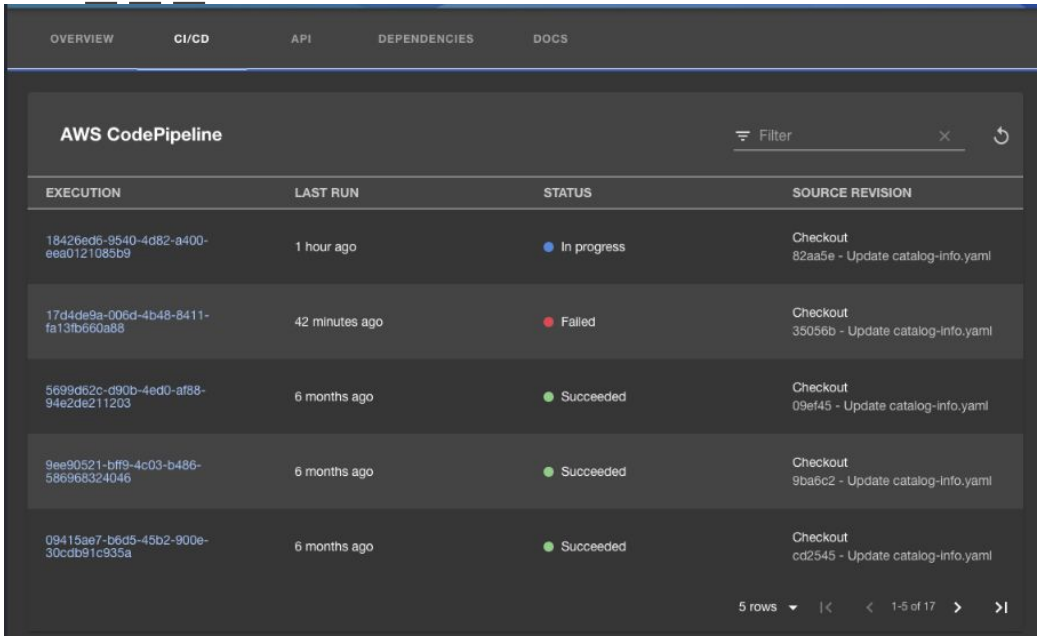
Pipeline config file look like this one :

The level of expression of config file is far more great than Backstage's Entity definition file

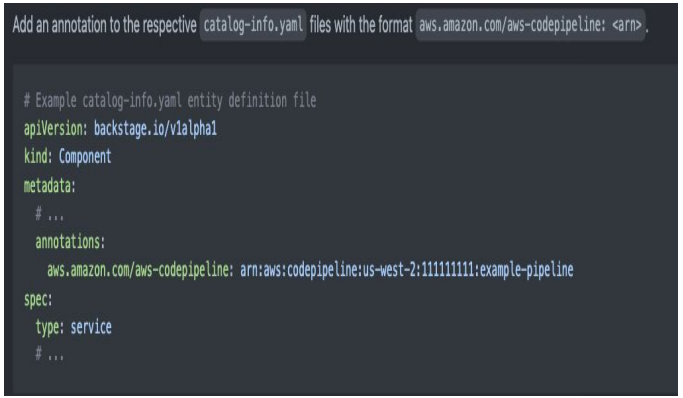
```
"service1" = {
  prefix                    = "service1"
  account_type              = "service"
  cpl_source_repository_id = "shannammtf/service1portal"
  cpl_source_branch_name    = "dev"
  sonar_project             = "shannammtf_service1portal"
  cb_env_environment_variable = [
    {
      name  = "ENV"
      type  = "PLAINTEXT"
      value = "dev"
    },
    {
      name  = "DISTRIBUTION_ID"
      type  = "PLAINTEXT"
      value = "E14WZHTMX5YQ6G"
    },
    {
      name  = "TARGET_ROLE_NAME"
      type  = "PLAINTEXT"
      value = "service1-int-cmn-codepipline-role"
    },
    {
      name  = "TARGET_ACCOUNT_ID"
      type  = "PLAINTEXT"
      value = "578789191732"
    }
  ]
  cpl_deploy_provider = "S3"
  cpl_deploy_role_arn = "arn:aws:iam::578789191732:role/service1-int-cmn-codepipline-role
  cpl_deploy_configuration = {
    BucketName = "service1-bucket"
    Extract    = true
  }
  enable_ecr = false
}
```

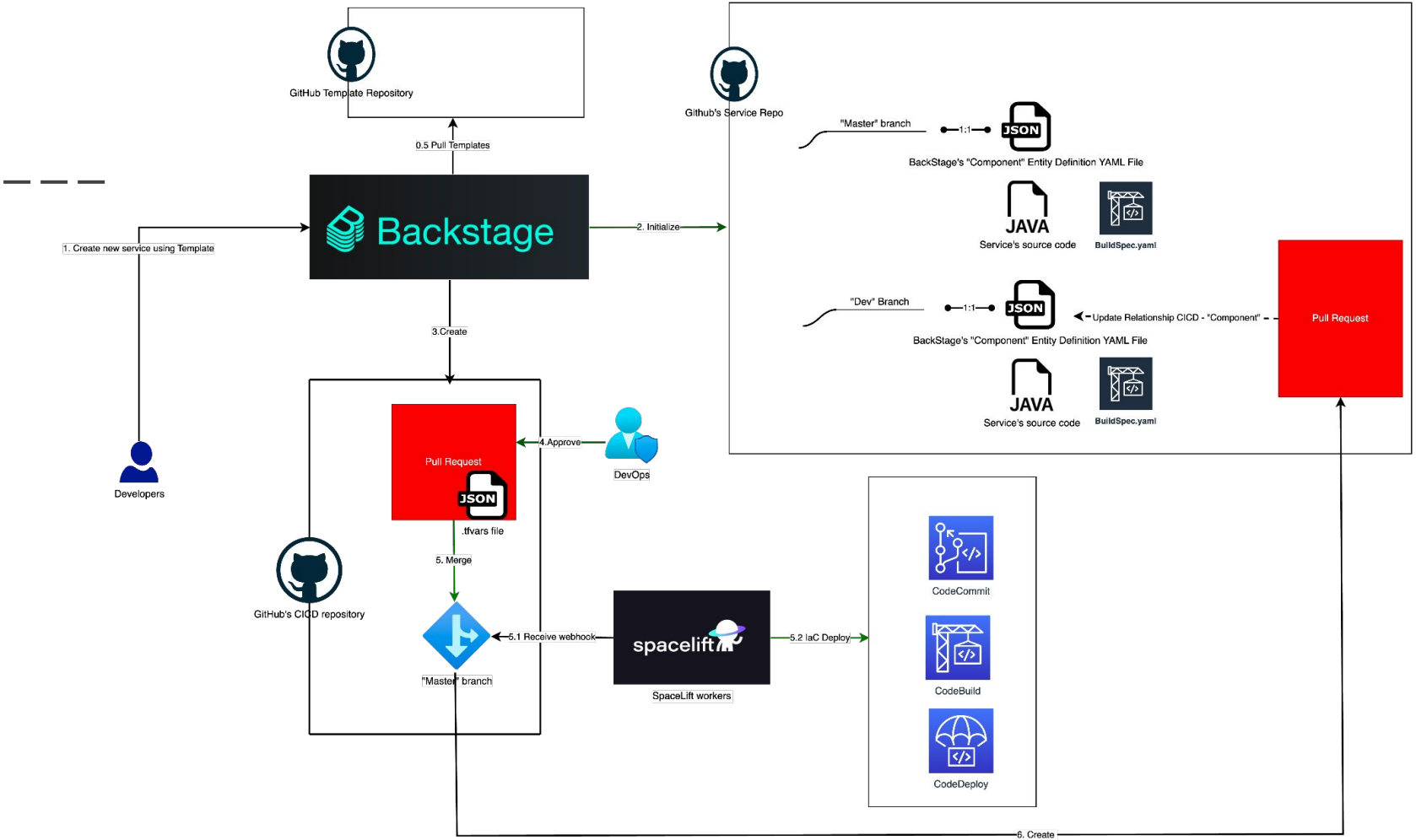
SpaceLift integration

AWS Code Services plugins for Backstage



After codepipeline deployed, we need to add its arn to “annotation” field of service’s entity definition file




GitHub Template Repository
0.5 Pull Templates
Backstage
1. Create new service using Template
2. Initialize
3.Create
Github's Service Repo
"Master" branch
1:1
JSON
BackStage's "Component" Entity Definition YAML File
JAVA
Service's source code
BuildSpec.yaml
"Dev" Branch
1:1
JSON
Update Relationship CICD - "Component"
BackStage's "Component" Entity Definition YAML File
JAVA
Service's source code
BuildSpec.yaml
Pull Request
Developers
Pull Request
JSON
4.Approve
DevOps
.tfvars file
5. Merge
GitHub's CICD repository
"Master" branch
5.1 Receive webhook
spacelift
SpaceLift workers
5.2 IaC Deploy
CodeCommit
CodeBuild
CodeDeploy
6. Create