

IAG Talents Day

Presenter : Shan Nam

Agenda

— — —

Introduce Myself

Self-service

Backstage core features:

- Service Catalog

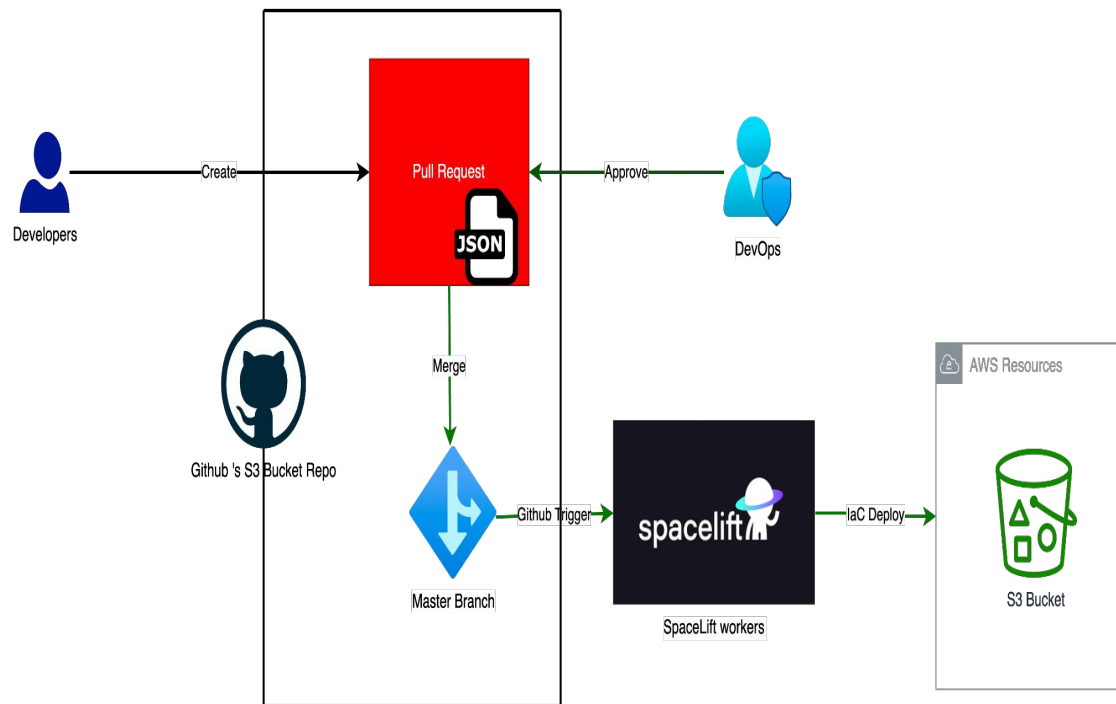
- Software Template

Backstage in real problem :

- Scenario 1 : Onboard 1st party-Service

- Scenario 2 : Provision AWS resources

Self-service



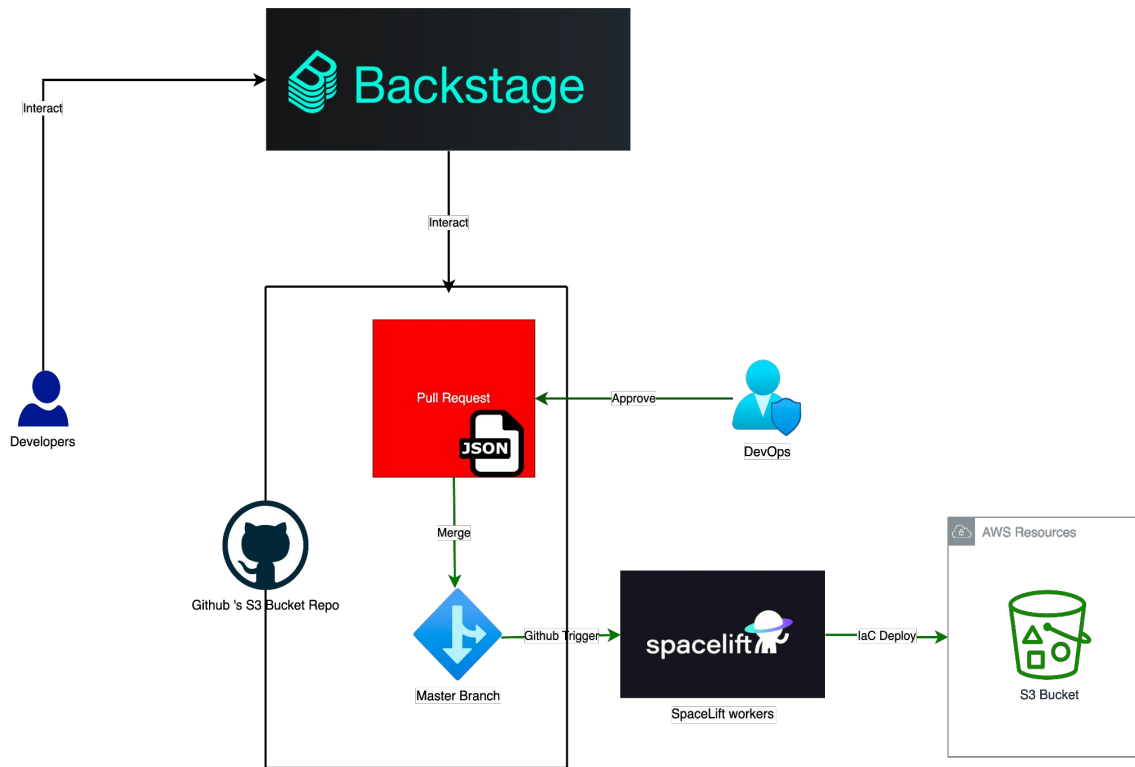
Disadvantages:

- 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

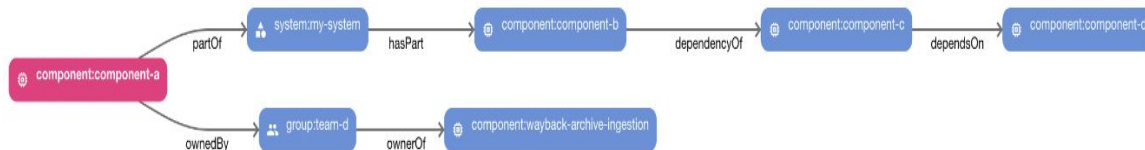
The screenshot displays the Datadog Service Catalog interface. On the left, a sidebar contains navigation options: 'Service Catalog', 'Explore', 'Setup & Config', and 'Scorecards'. Below these are tabs for 'List' and 'Map', and a search bar. The main content area is divided into three sections: 'Ownership' (Find Service Owners), 'Reliability' (Evaluate Operational Health), and 'Performance' (Monitor Application Performance). The 'Performance' section is active, showing a list of services under 'My Teams'. The 'web-store' service is selected, and its details are shown on the right. The 'Service Metadata' section includes a table with columns: TEAM (Shopist), APPLICATION (shopist), LIFECYCLE (production), and TIER (tier1). Below this, there are sections for 'ON-CALL' (via Pagerduty), 'CODE REPOSITORIES' (Deployment, Source), 'DOCUMENTATION' (Deployment Information, Service Documentation), 'CONTACTS' (Open in slack), 'RUNBOOKS' (Common Operations, Disabling Deployments, Rolling Back Deployments), and 'DASHBOARD LINKS' (Demo Dashboard). At the bottom, there is a 'Setup Guidance' section with tabs for 'Dashboards', 'Definition', 'Dependencies', 'Tracing Configuration', 'Libraries', and 'Scorecards'. The 'Definition' tab is active, showing 'Finish setting up your service' and 'Recommended setup for this service' with a list of recommended actions and their status (DETECTED).

Service Catalog : From Backstage

Provide useful information to Dev team :

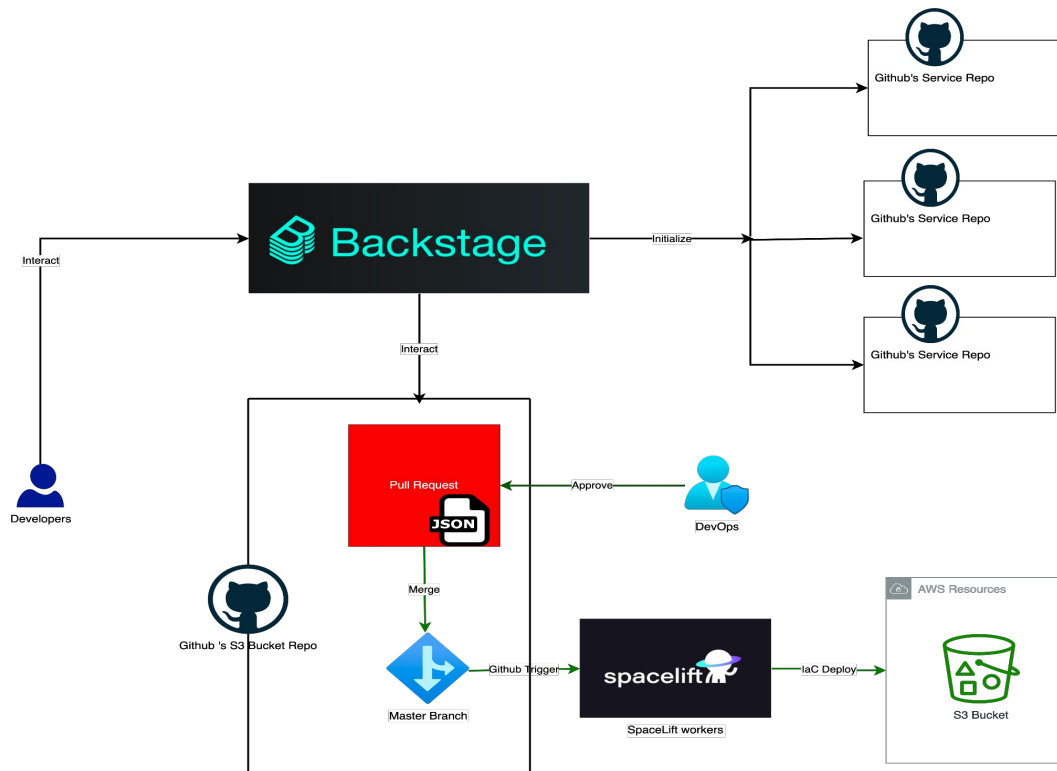
- Ownership, Pipeline's status, OpenAPI Visualization, Docs.
- More meaningful relationships in Dependencies graph : App-App, App-resources, App-user/group
- Sandbox-like capacity: Can be customized, even import Metrics dashboard from, for example, Datadog

All components (1) Filter ×							
NAME	SYSTEM	OWNER	TYPE	LIFECYCLE	DESCRIPTION	TAGS	ACTIONS
springboot-payment		dev-team	service	experimental	payment service...		



OVERVIEW	CI/CD	API	DEPENDENCIES	DOCS
AWS CodePipeline Filter × ↺				
EXECUTION	LAST RUN	STATUS		SOURCE REVISION
18426ed6-9540-4d82-a400- eea0121085b9	1 hour ago	● In progress		Checkout 82aa5e - Update catalog-info.yaml

Software template



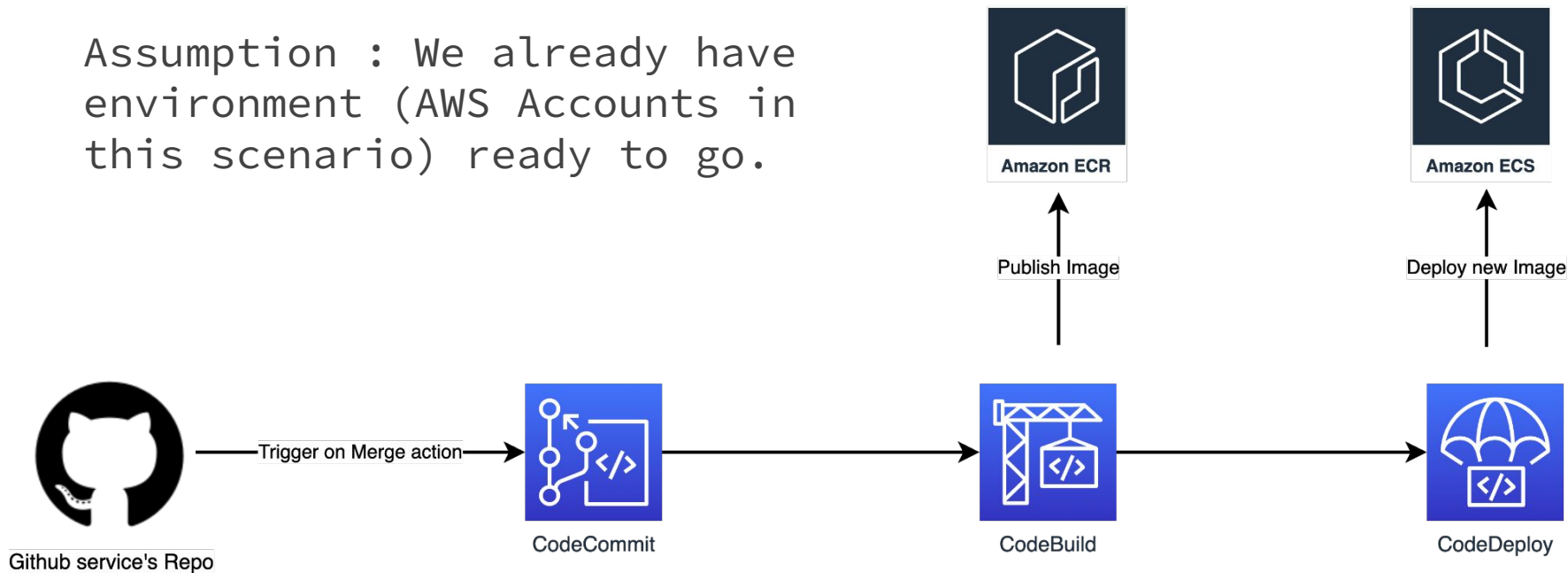
Not be limited to “template engine”

- 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.

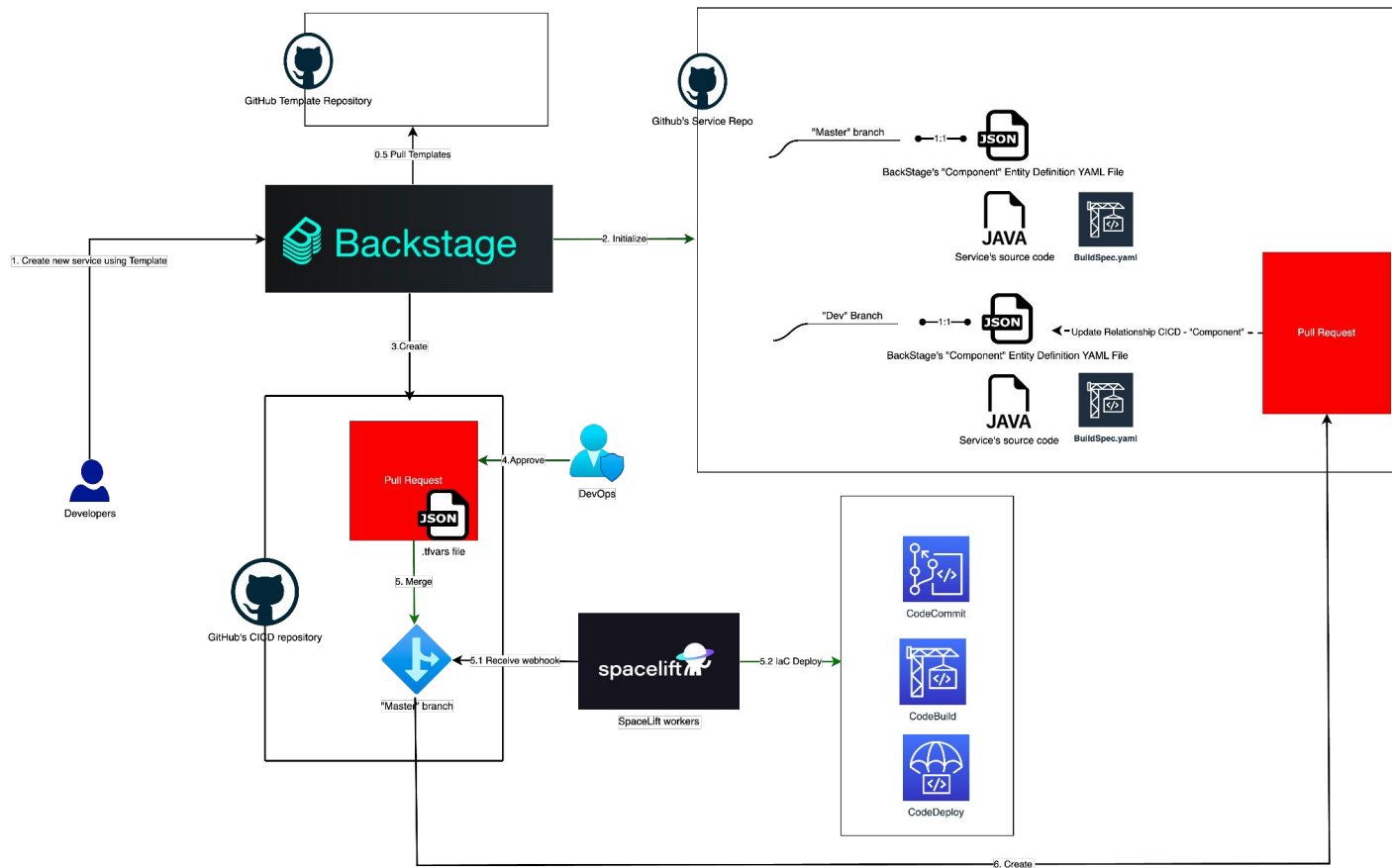


Scenario 1 : Onboard 1st party-Service

Requirements:

- Single Entry-point of onboard new Service :
Backstage-Template
- The Result we would like to get :
 - New Repository for service : Master, Dev branches
 - CI/CD 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 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

→ github.com/orgs/shannamtf/people

shannamtf

Overview Repositories 4 Projects Packages Teams 1 People 2 Settings

GitHub users are [now required](#) to enable two-factor authentication as an additional security measure. Your activity on GitHub includes you in this requirement. You will be required to enable two-factor authentication by September 14, 2023, or be restricted from account actions.

Organization permissions ⓘ

Members 2

Outside collaborators

Pending collaborators

Invitations

Failed invitations

You are the only owner of this organization! We recommend a minimum of two

☐ Members

☐ Chuym Mee Tinh Nhựt
gimmetinhnhua

☐ gimmetinhyeu

New team

Select all

Visibility ▾ Members ▾

☐ dev-team

Dev team

2 members 0 roles 0 teams

GROUP — TEAM

dev-team ☆

OVERVIEW

dev-team

Dev team

N/A
Parent Group

N/A
Child Groups

Members (2)

of dev-team

Chuym Mee Tinh Nhựt

gimmetinhyeu

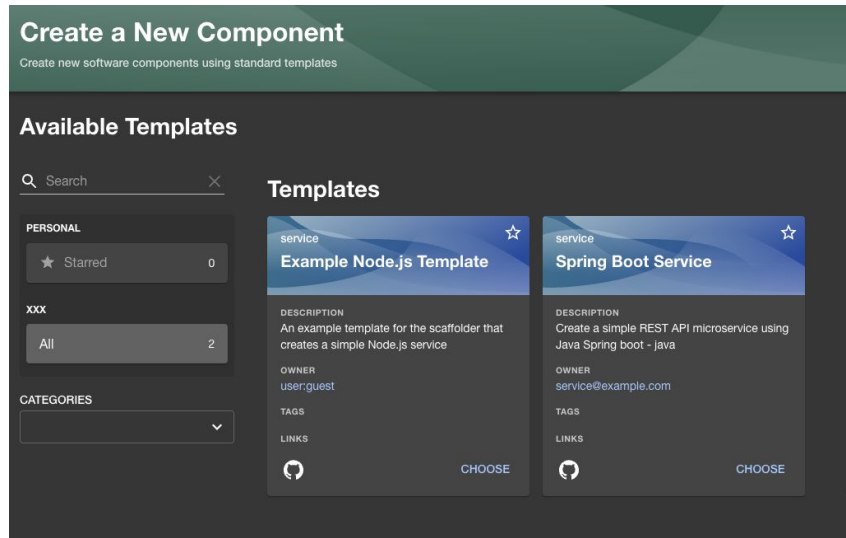
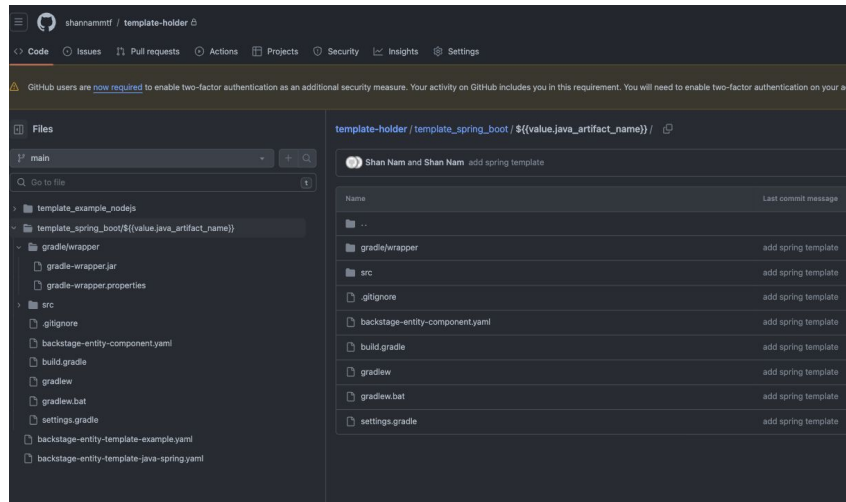
Step 0.5 A Dedicated Repository contains different kinds of templates

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

1  apiVersion: scaffolder.backstage.io/v1beta3
2  kind: Template
3  metadata:
4    name: springboot-template
5    title: Spring Boot Service
6    description: Create a simple REST API microservice using Java Spring boot
7    - java
8  spec:
9    owner: service@example.com
10   type: service
11
12   parameters:
13     - title: Provide some simple information
14       required:
15         - component_id
16         - owner
17         - java_package_name
18         - description
19   properties:
20     component_id:
21       title: Name
22       type: string
23       description: Unique name of the component
24       ui:field: EntityNamePicker
25     java_package_name:
26       title: Java Package Name
27       type: string
28       description: Name for the java package. eg (payment)
29     description:
30       title: Description
31       type: string
32       description: Help others understand what this website is for.
33   owner:
34     title: Owner
35     type: string
36     description: Owner of the component
37     ui:field: OwnerPicker
38     ui:options:
39       allowedKinds:
40         - Group
```



Step 1: Dev creates new Service base on predefined Template

Create a New Component

Create new software components using standard templates

Spring Boot Service

1

Provide some simple information

Name*

springboot-payment

Unique name of the component

Java Package Name*

springbootpayment

Name for the java package. eg (payment)

Description*

payment service created from java spring boot template

Help others understand what this website is for.

Owner*

dev-team

Owner of the component

BACK

NEXT STEP

2

Choose a location

Create a New Component

Create new software components using standard templates

Spring Boot Service

✓

Provide some simple information

2

Choose a location

Host

github.com

▼

The host where the repository will be created

Owner*

shannammf

The organization, user or project that this repo will belong to

Repository*

springboot-payment

The name of the repository

BACK

NEXT STEP

14

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

Task Activity

Activity for task: e727f747-6c18-47a3-a4fe-8d202103928b

- ✓ Fetch Skeleton + Template 2 seconds
- ✓ Publish 5 seconds
- ✓ pr 5 seconds

🌐 Repository

START OVER

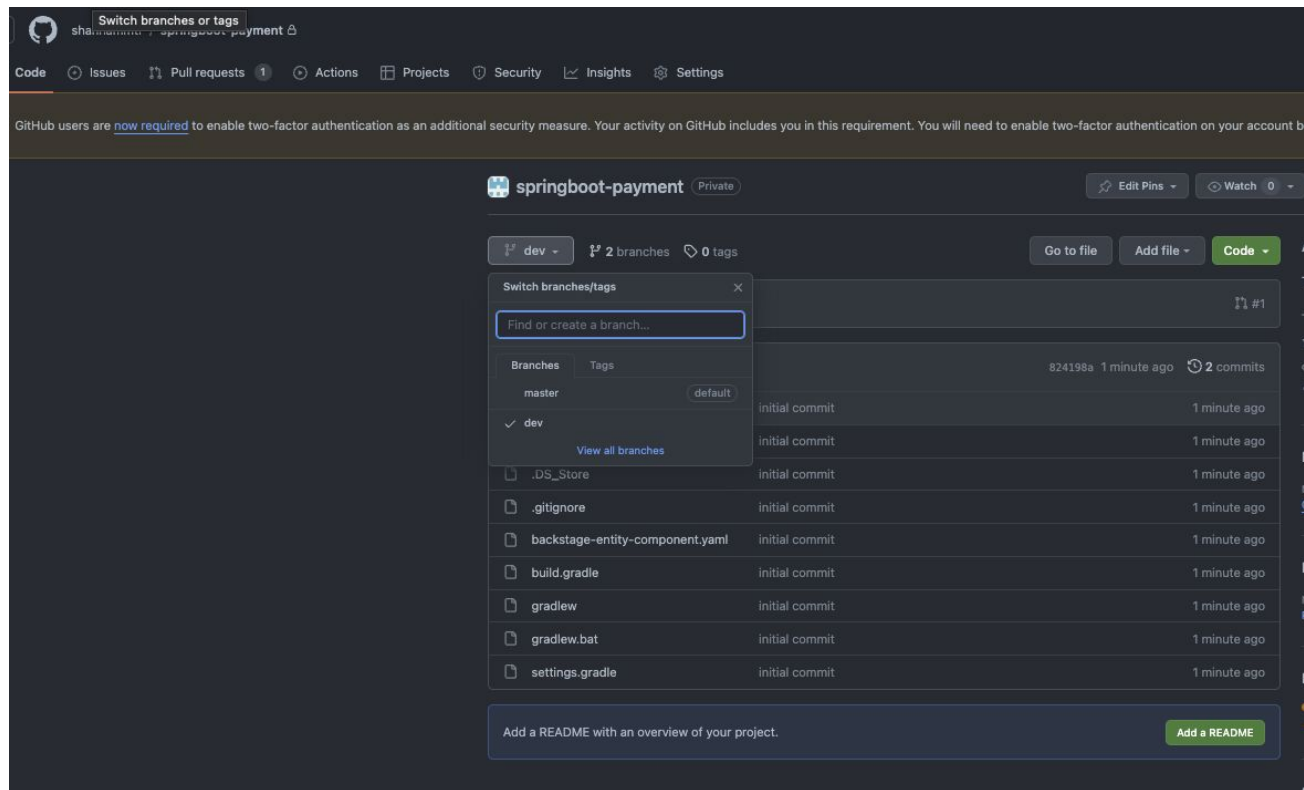
CANCEL

```
1 2023-11-20T16:49:07.000Z Beginning step pr
2 2023-11-20T16:49:12.000Z Finished step pr
```

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:

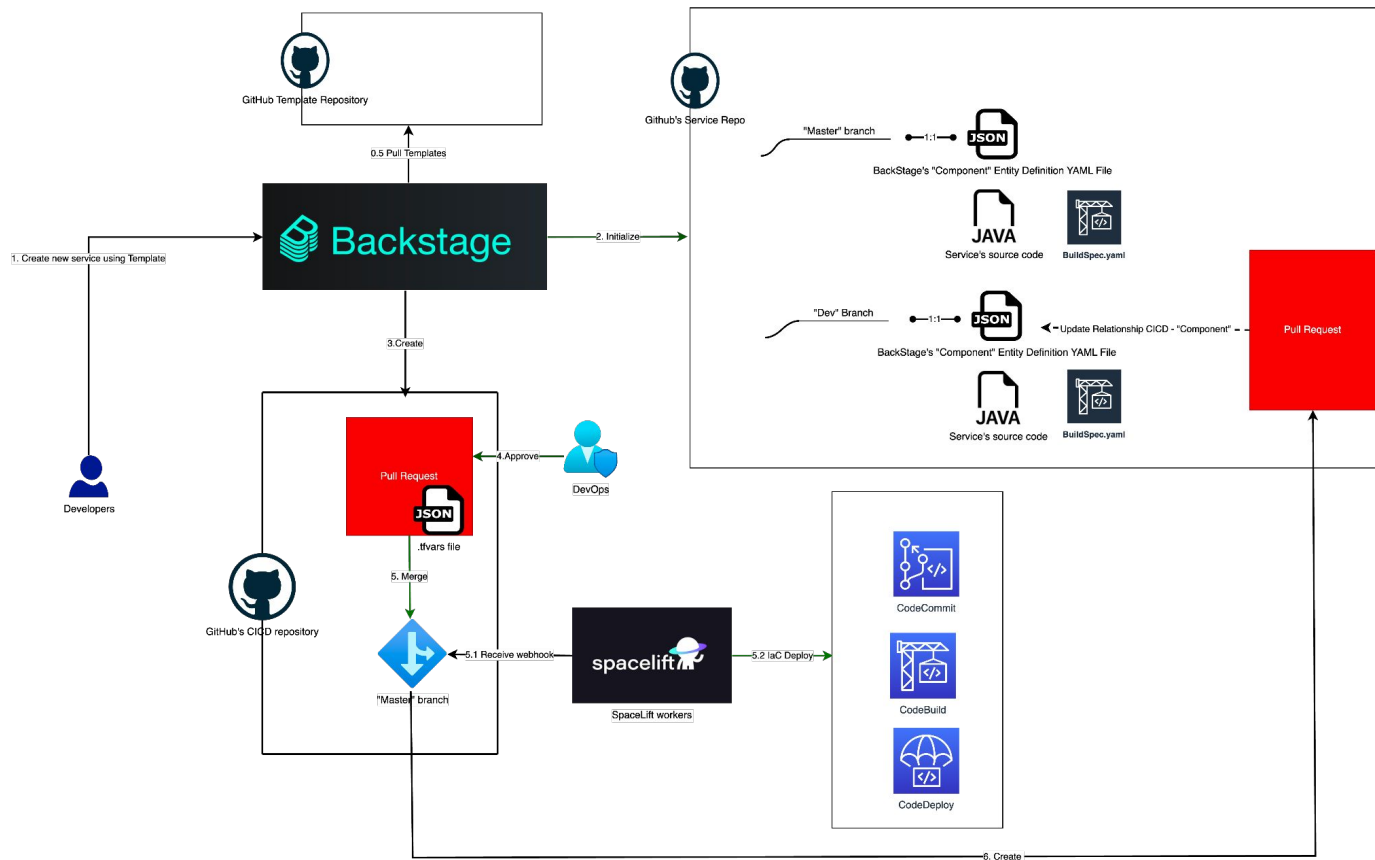
- Fetch template, create new source code for service based on param
- Create new Repo with single “master” branch
- Create a PR from “dev” branch to “master” branch which results in creating new “dev” branch

```
steps:
  - id: template
    name: Fetch Skeleton + Template
    action: fetch:template
    input:
      url: https://github.com/shannamtmf/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 }

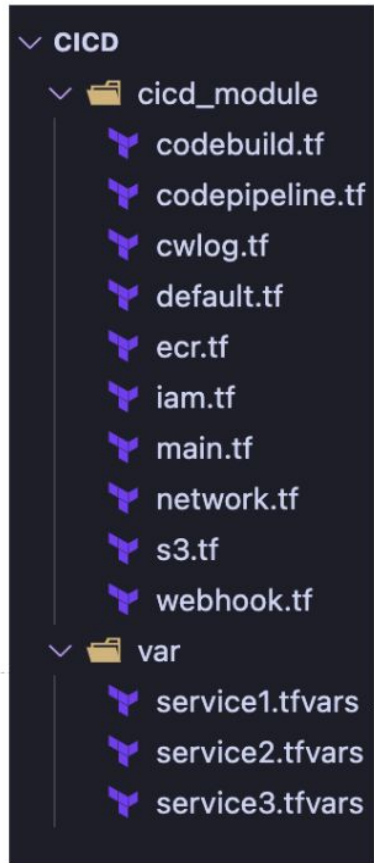
  - id: pr_create_branch
    name: pr
    action: publish:github:pull-request
    input:
      repoUrl: ${ parameters.repoUrl }
      branchName : dev
      targetBranchName : master
      title: to create new branch
      description : to create new branch
```

Workflow:



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.

←Add new .tfvars file to "var" folder→

Pull Request

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 = "shannamtf/serviceportal"
  cpl_source_branch_name = "dev"
  sonar_project = "shannamtf_serviceportal"
  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-codepipeline-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-codepipeline-role"
  cpl_deploy_configuration = {
    BucketName = "service1-bucket"
    Extract = true
  }
  enable_ecr = false
}
```

SpaceLift integration

— — —

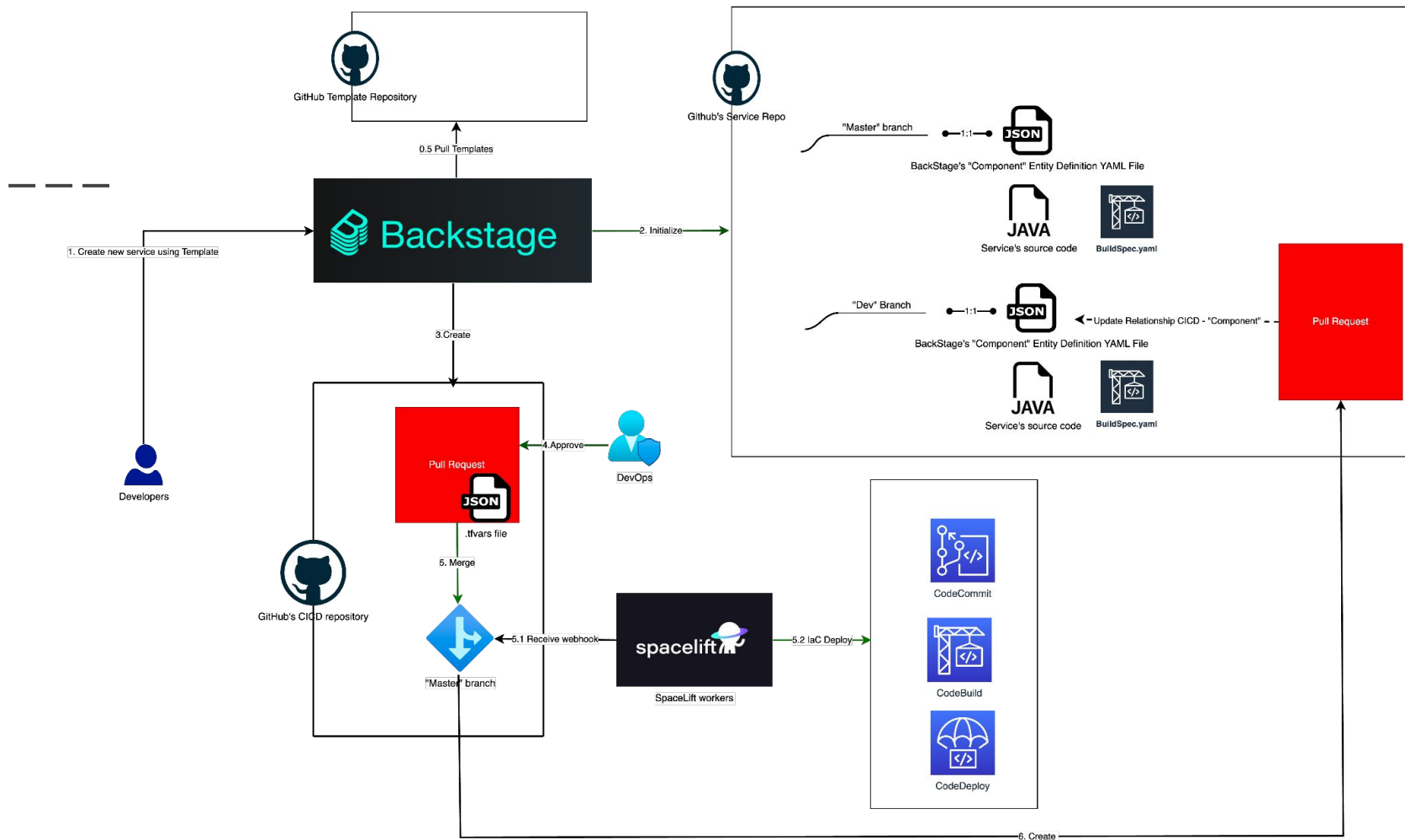
AWS Code Services plugins for Backstage

OVERVIEW	CI/CD	API	DEPENDENCIES	DOCS
AWS CodePipeline				
EXECUTION	LAST RUN	STATUS	SOURCE REVISION	
18426ed6-9540-4d82-a400- eea0121085b9	1 hour ago	In progress	Checkout 82aa5e - Update catalog-info.yaml	
17d4de9a-006d-4b48-8411- fa13fb660a88	42 minutes ago	Failed	Checkout 3505eb - Update catalog-info.yaml	
5699d62c-d90b-4ed0-af88- 94e2de211203	6 months ago	Succeeded	Checkout 09ef45 - Update catalog-info.yaml	
9ee90521-bff9-4c03-b486- 586968324046	6 months ago	Succeeded	Checkout 9ba6c2 - Update catalog-info.yaml	
09415ae7-b6d5-45b2-900e- 30cdb91c935a	6 months ago	Succeeded	Checkout cd2545 - Update catalog-info.yaml	
5 rows 1-5 of 17				

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

Add an annotation to the respective `catalog-info.yaml` files with the format `aws.amazon.com/aws-codepipeline: <arn>`.

```
# Example catalog-info.yaml entity definition file
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  # ...
  annotations:
    aws.amazon.com/aws-codepipeline: arn:aws:codepipeline:us-west-2:111111111:example-pipeline
spec:
  type: service
  # ...
```



— — —