

PORT SUPPPORT ENGINEER ASSIGNMENT

Candidate: Asomugha Divine

Date: 11/08/2025

Table of Contents

1. Exercise #1 – JQ Patterns
2. Exercise #2 – Jira & GitHub Integration
3. Exercise #3 – PR Count Scorecard
4. Exercise #4 – GitHub Workflow Troubleshooting

Exercise #1 – JQ Patterns.

1(a) Extract current replica count

```
```.spec.replicas``
```

**Explanation:** `.spec.replicas` points to the integer value of the current replica count in a Kubernetes Deployment object.

```
divine_ubuntu@DESKTOP-FLD95D5:~$ jq '. | keys' k8s-deploy.json
[
 "apiVersion",
 "kind",
 "metadata",
 "spec",
 "status"
]
divine_ubuntu@DESKTOP-FLD95D5:~$ jq '.spec | keys' k8s-deploy.json
[
 "progressDeadlineSeconds",
 "replicas",
 "revisionHistoryLimit",
 "selector",
 "strategy",
 "template"
]
divine_ubuntu@DESKTOP-FLD95D5:~$ jq '.spec.replicas' k8s-deploy.json
1
```

The screenshot shows the JQ Playground interface. The 'QUERY' tab is active, displaying the query `1 .spec.replicas`. The 'OUTPUT' tab shows the result `1 1`. Below the query, there is an 'Options' dropdown menu. At the bottom, the 'JSON' tab is active, showing the full JSON structure of the Kubernetes Deployment object. The 'HTTP' tab is also visible. A 'Report a Bug' button is located in the bottom right corner.

QUERY

```
1 .spec.replicas
```

Options

JSON HTTP

```
1 {
2 "apiVersion": "apps/v1",
3 "kind": "Deployment",
4 "metadata": {
5 "annotations": {
6 "deployment.kubernetes.io/revision": "1",
7 "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"app
8 },
9 "creationTimestamp": "2023-07-27T14:41:15Z",
10 "generation": 2,
11 "labels": {
12 "commitHash": "b2633eae0655322b22f1637eb309ba4052ceeb74",
13 "environment": "production-gcp-1",
14 "service": "authorization"
15 },
16 "name": "authorization-service-newproduction-0001"
```

OUTPUT

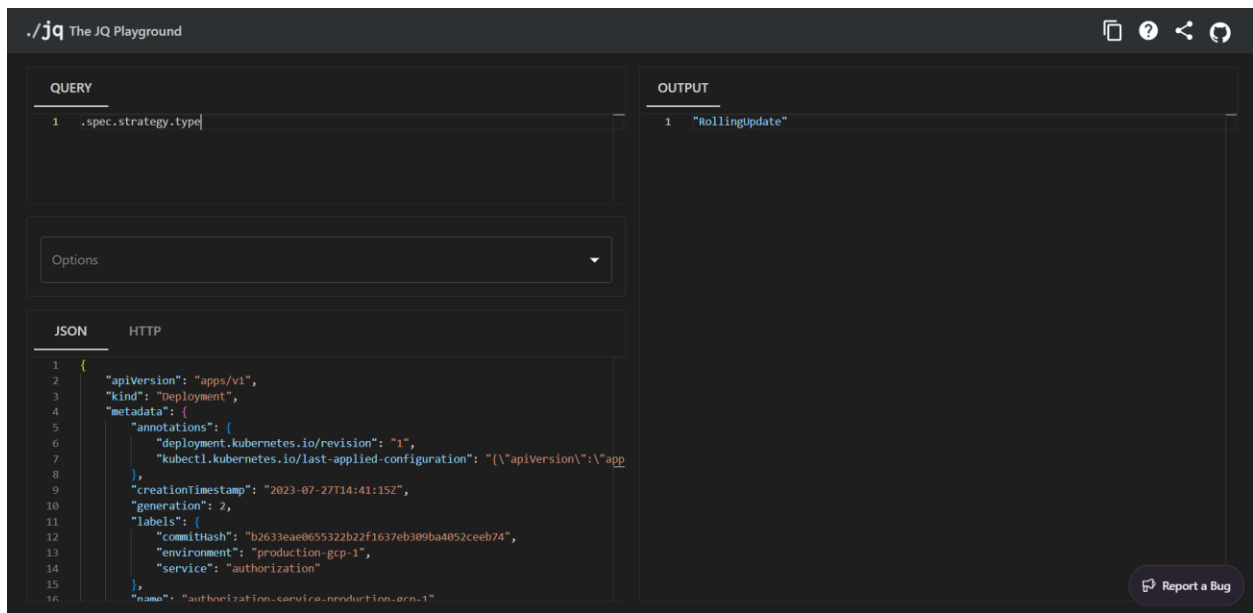
```
1 1
```

Report a Bug

### 1(b) Extract deployment strategy

```
```spec.strategy.type```
```

Explanation: The deployment strategy type (e.g., RollingUpdate or Recreate) is found under `.spec.strategy.type`.



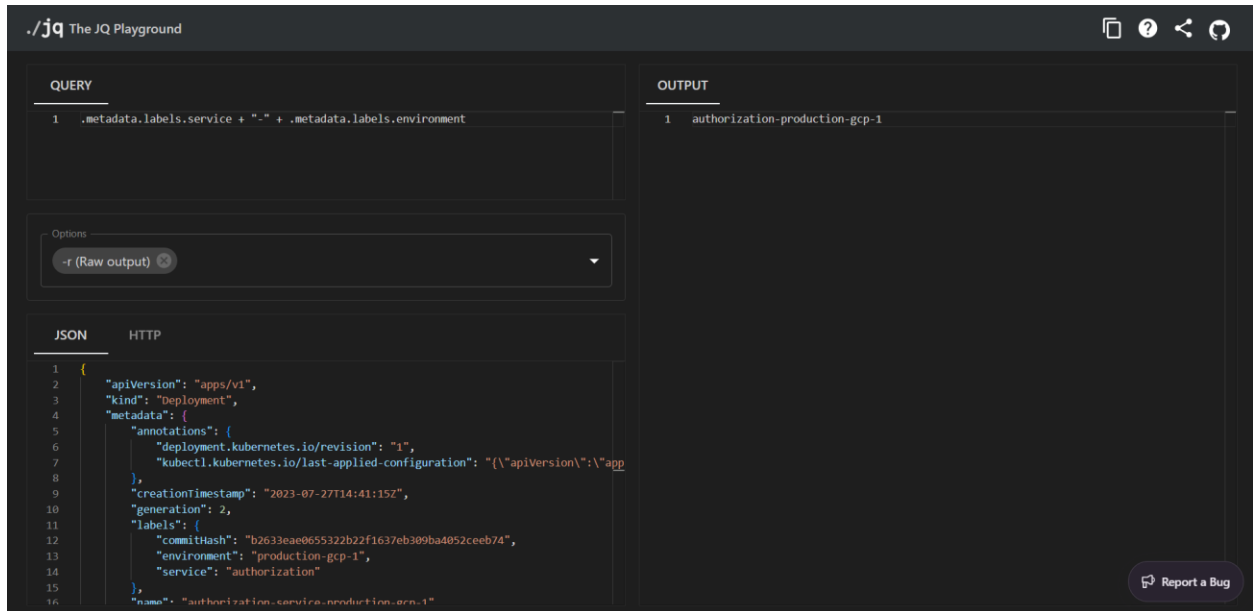
The screenshot shows the 'jq The JQ Playground' interface. In the 'QUERY' section, the query `.spec.strategy.type` is entered. The 'OUTPUT' section displays the result `"RollingUpdate"`. Below the query, there is a 'JSON' section showing a sample deployment object. The 'HTTP' tab is also visible.

```
1 {
2   "apiVersion": "apps/v1",
3   "kind": "Deployment",
4   "metadata": {
5     "annotations": {
6       "deployment.kubernetes.io/revision": "1",
7       "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"app
8     },
9     "creationTimestamp": "2023-07-27T14:41:15Z",
10    "generation": 2,
11    "labels": {
12      "commitHash": "b2633eae0655322b22f1637eb309ba4052ceeb74",
13      "environment": "production-gcp-1",
14      "service": "authorization"
15    },
16    "name": "authorization-service-production-001"
```

1(c) Extract the “service” label of the deployment concatenated with the “environment” label of the deployment, with a hyphen (-) in the middle.

```
```metadata.labels.service + "-" + .metadata.labels.environment```
```

**Explanation:** Accesses the two label values and concatenates them into a single string.



## Image proof from local pc

```
divine_ubuntu@DESKTOP-FLD95D5:~$ jq '.status.replicas' k8s-deploy.json
1
divine_ubuntu@DESKTOP-FLD95D5:~$ jq '.spec.strategy.type' k8s-deploy.json
"RollingUpdate"
divine_ubuntu@DESKTOP-FLD95D5:~$ jq -r '.metadata.labels.service + "-" + .me
tadata.labels.environment' k8s-deploy.json
authorization-production-gcp-1
divine_ubuntu@DESKTOP-FLD95D5:~$ |
```

## 2- Extract all the issue IDs (for example SAMPLE-123) for all subtasks, in an array.

```
```.fields.subtasks[].key```
```

Explanation: `.fields.subtasks` is an array of subtasks; selecting `.key` from each returns their IDs. Wrapping with `[...]` ensures an array output:

./jq The JQ Playground

QUERY

```
1 [.fields.subtasks[].key]
```

Options

JSON HTTP

```
6 "fields": {
2884   "reporter": {
2895     "timeZone": "Asia/Jerusalem",
2896     "accountType": "atlassian"
2897   },
2898   "customfield_10000": "()",
2899   "aggregateprogress": {
2900     "progress": 0,
2901     "total": 0
2902   },
2903   "customfield_10001": null,
2904   "customfield_10002": null,
2905   "customfield_10003": null,
2906   "customfield_10004": null,
2907   "environment": null,
2908   "duedate": null,
```

OUTPUT

```
1 [
2   "SAMPLE-3894",
3   "SAMPLE-3895",
4   "SAMPLE-3896",
5   "SAMPLE-3897",
6   "SAMPLE-3898",
7   "SAMPLE-3899",
8   "SAMPLE-3900",
9   "SAMPLE-3902",
10  "SAMPLE-3904",
11  "SAMPLE-3901",
12  "SAMPLE-3905",
13  "SAMPLE-3906",
14  "SAMPLE-3907"
15 ]
```

Report a Bug

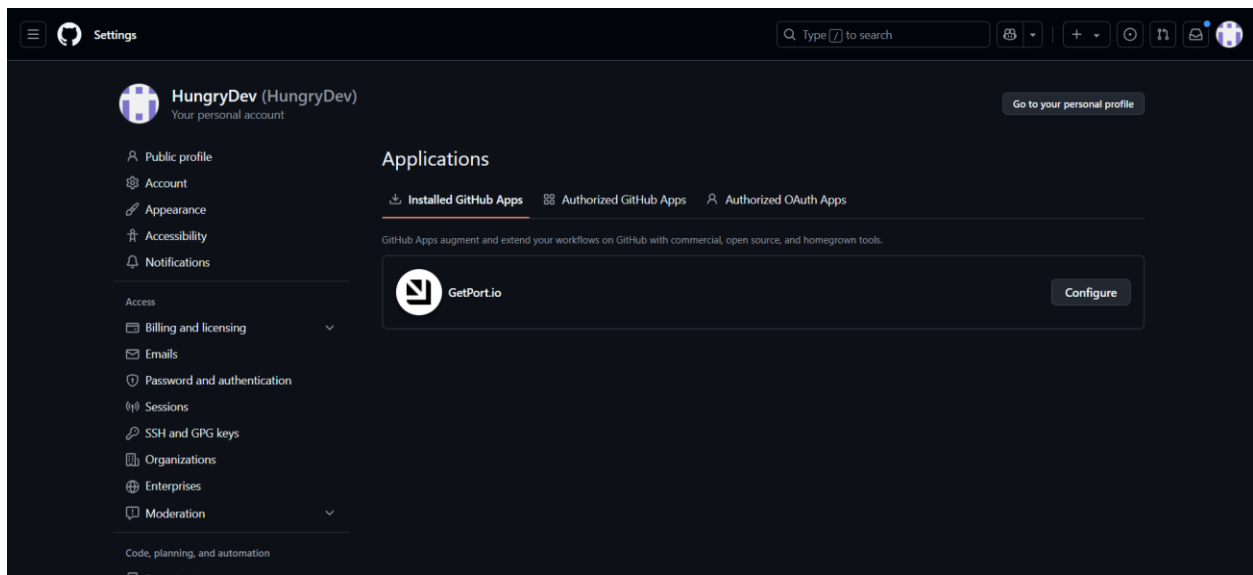
```
divine_ubuntu@DESKTOP-FLD95D5:~$ jq '[.fields.subtasks[].key]' jira-api-issu
e.json
[
  "SAMPLE-3894",
  "SAMPLE-3895",
  "SAMPLE-3896",
  "SAMPLE-3897",
  "SAMPLE-3898",
  "SAMPLE-3899",
  "SAMPLE-3900",
  "SAMPLE-3902",
  "SAMPLE-3904",
  "SAMPLE-3901",
  "SAMPLE-3905",
  "SAMPLE-3906",
  "SAMPLE-3907"
]
divine_ubuntu@DESKTOP-FLD95D5:~$ |
```

Exercise #2 – Jira & GitHub Integration.

Steps Taken:

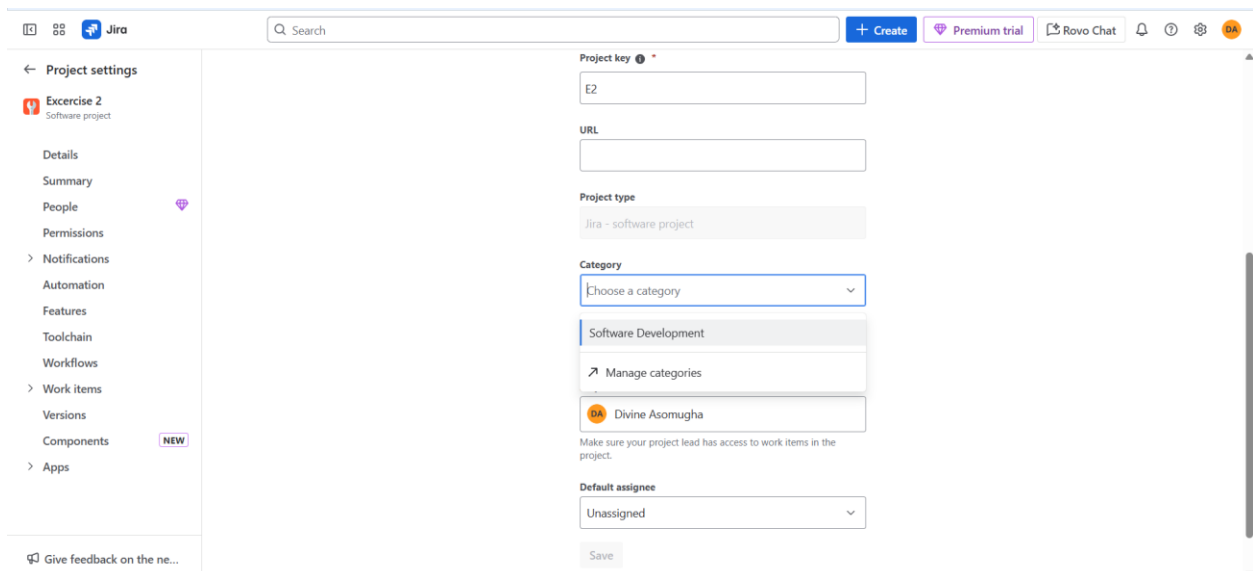
1. Signed Up to Port with this email. divine+port@solmateai.site
2. Installed Port's GitHub app and authorized repos.

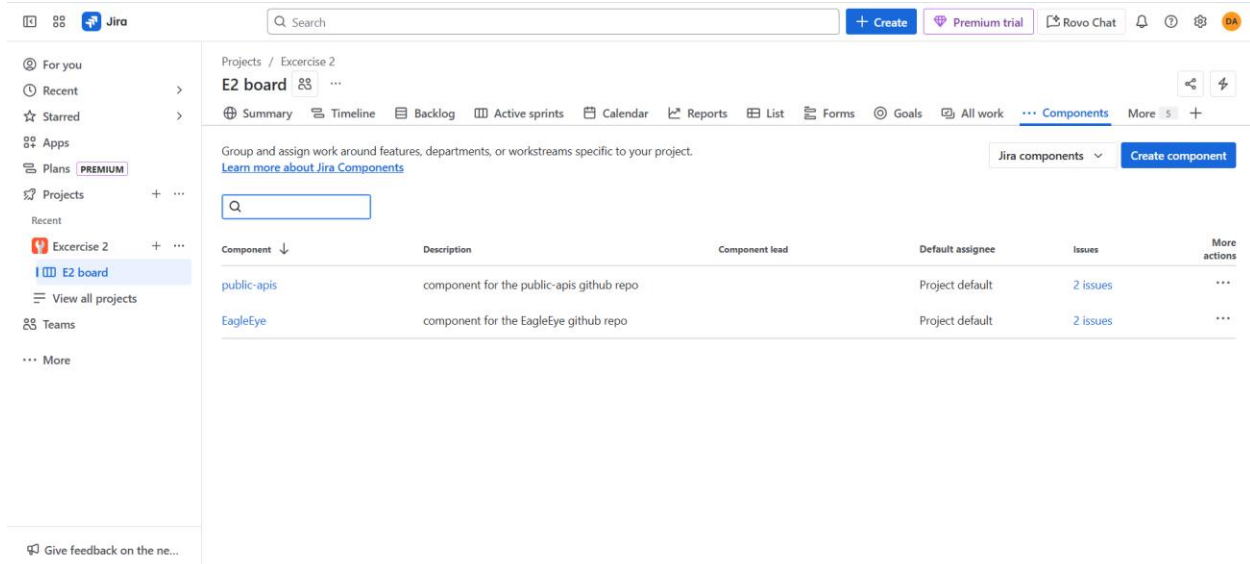
Image below shows the installed getport github application



3. Created Jira project (Exercise2) with Scrum template (company-managed).

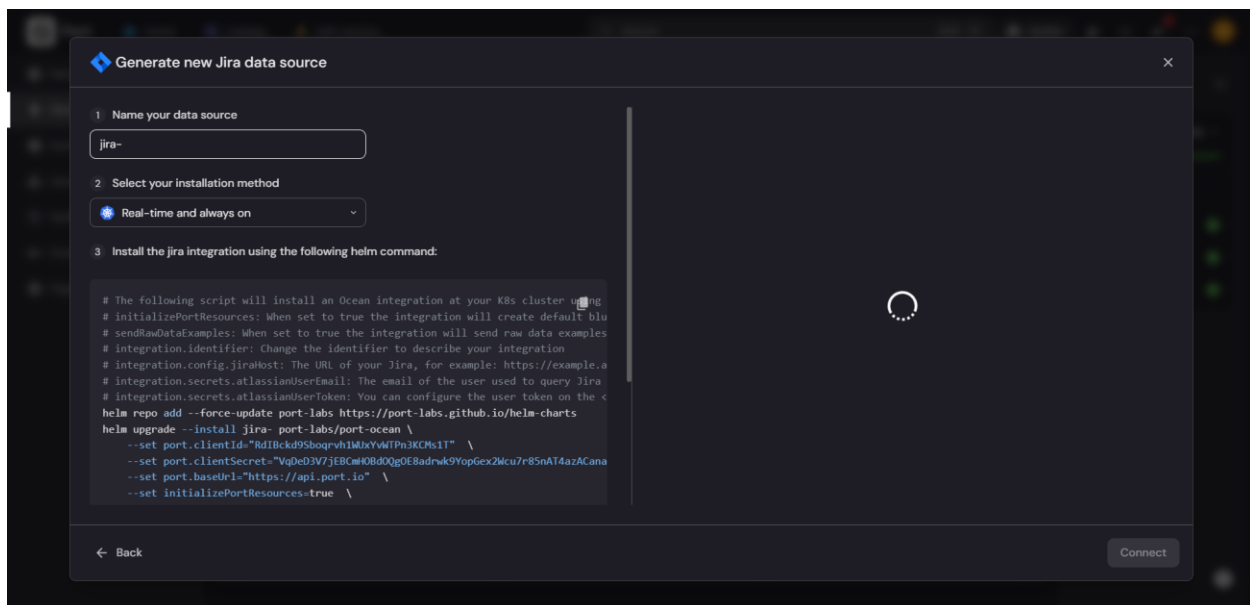
Images below shows the created Jira project





4. Installed Port Ocean Integration for Jira via Helm.

Image below shows the installation instruction for Port Ocean Integration for Jira



```
helm repo add --force-update port-labs https://port-labs.github.io/helm-charts
helm upgrade --install jira port-labs/port-ocean \
--set port.clientId="RdIBckd9SboqrVh1WUXyVWPn3KCMs1T" \
--set port.clientSecret="VqDeD3v7jEBcMHOBD0QgOE8adrwk9YopGex2Wcu7r85nAT4azACanaaThOtVh03x" \
--set port.baseUrl="https://api.port.io" \
--set initializePortResources=true \
--set sendRawDataExamples=true \
--set scheduledResyncInterval=360 \
--set integration.identifier="jira" \
--set integration.type="jira" \
--set integration.eventListener.type="POLLING" \
--set integration.config.jiraHost="https://hyperfloxx.atlassian.net" \
--set integration.secrets.atlassianUserEmail="divine@solmateai.site" \
--set integration.secrets.atlassianUserToken="ATATT3xFGF0tInVRG-dzzI_3ejl4B-
io4uywZwSaFciCz_MDZMB2q5VaygaK1_bnefffffdedddddddeeeeeeefewDxonJXntUtuJ2eZQ1vzaPn5JjhjuFqQSqE9dQPm87viVmk9SEBD4f
v4CMEk3-pwsPMTjYkt4Xw=F210E987"
```

Cluster

Nodes

Workloads

Overview

Pods

Deployments

DaemonSets

StatefulSets

ReplicaSets

Replication Controllers

Jobs

CronJobs

Config

Network

Storage

Namespaces

Events

Helm

Charts

Releases

Access Control

Custom Resources

Overview

Pods

Deployments

DaemonSets

StatefulSets

ReplicaSets

Replication Controllers

Jobs

CronJobs

Pods

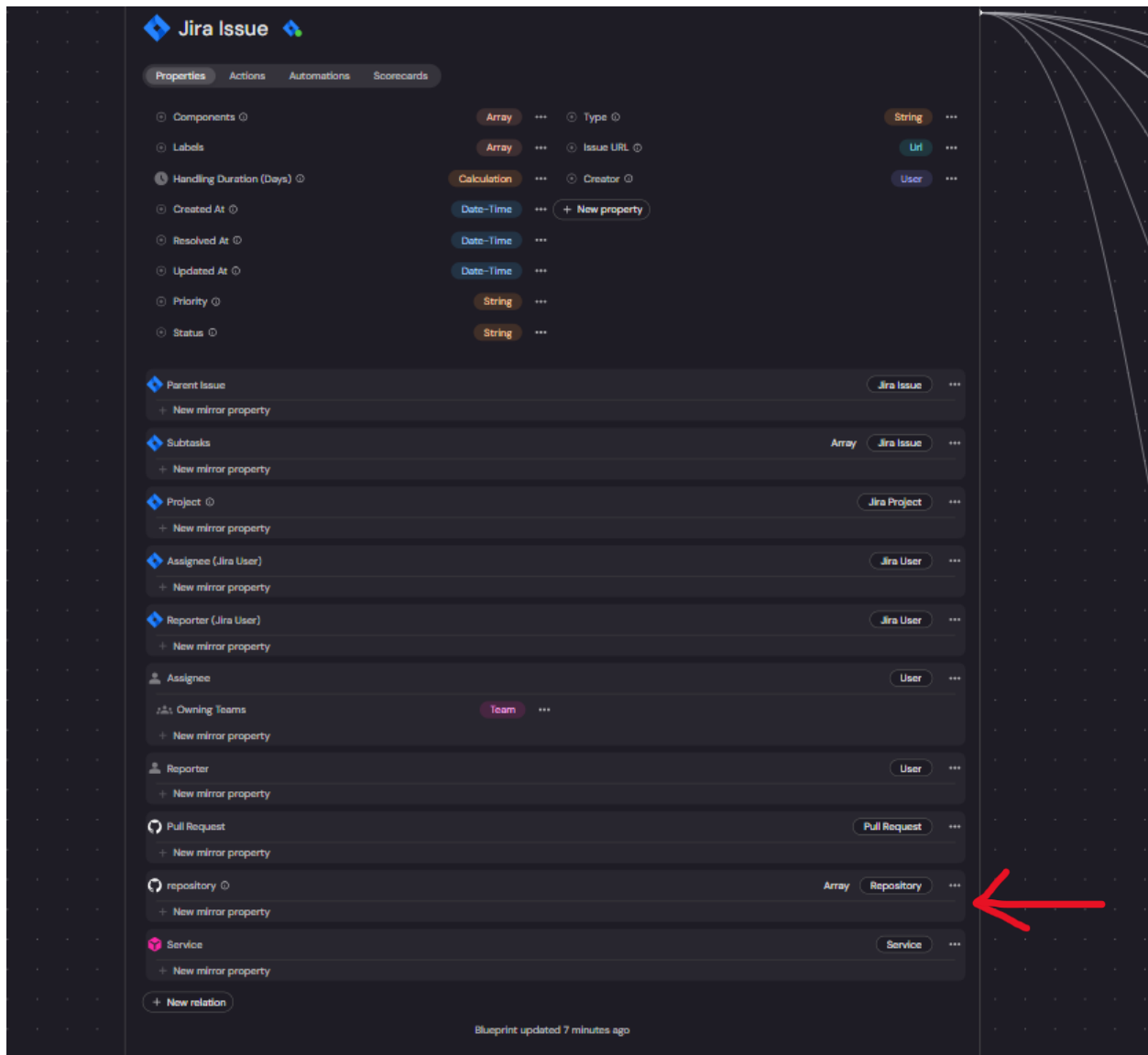
1 item

Namespace: default

Search Pods...

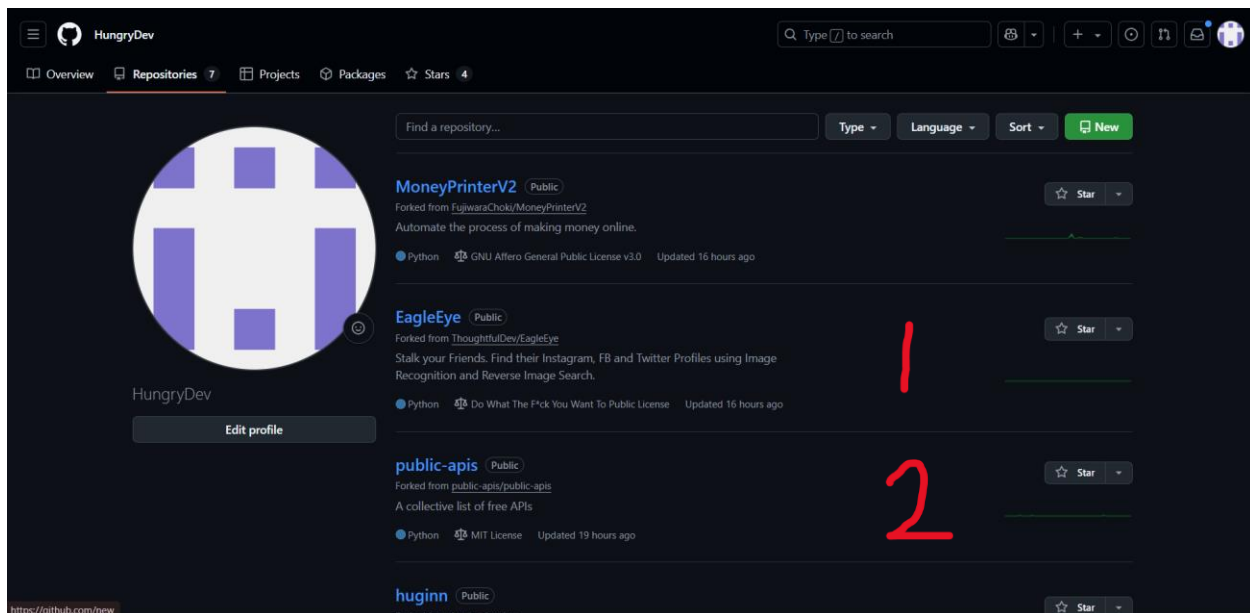
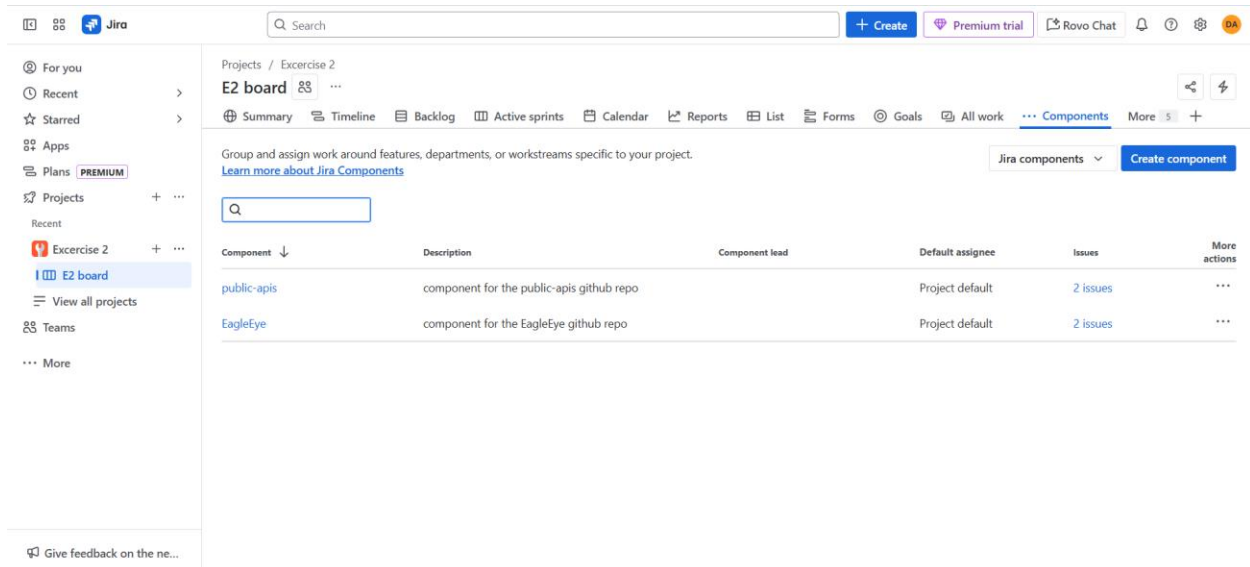
	Name	Namespace	Containers	Restarts	Controlled By	Node	QoS	Age	Status	
<input type="checkbox"/>	ocean-jira-jira-deployment-6778r	default	<div></div>	1	ReplicaSet	docker-desktop	Burstable	22h	Running	

Image below shows the added relation btw Jira issues and GitHub repositories



6. Created matching Jira components for repos.

Images below show the components and the repositories with matching names



7. Updated Jira integration mapping.

Image below shows the Jira issue updated mapping

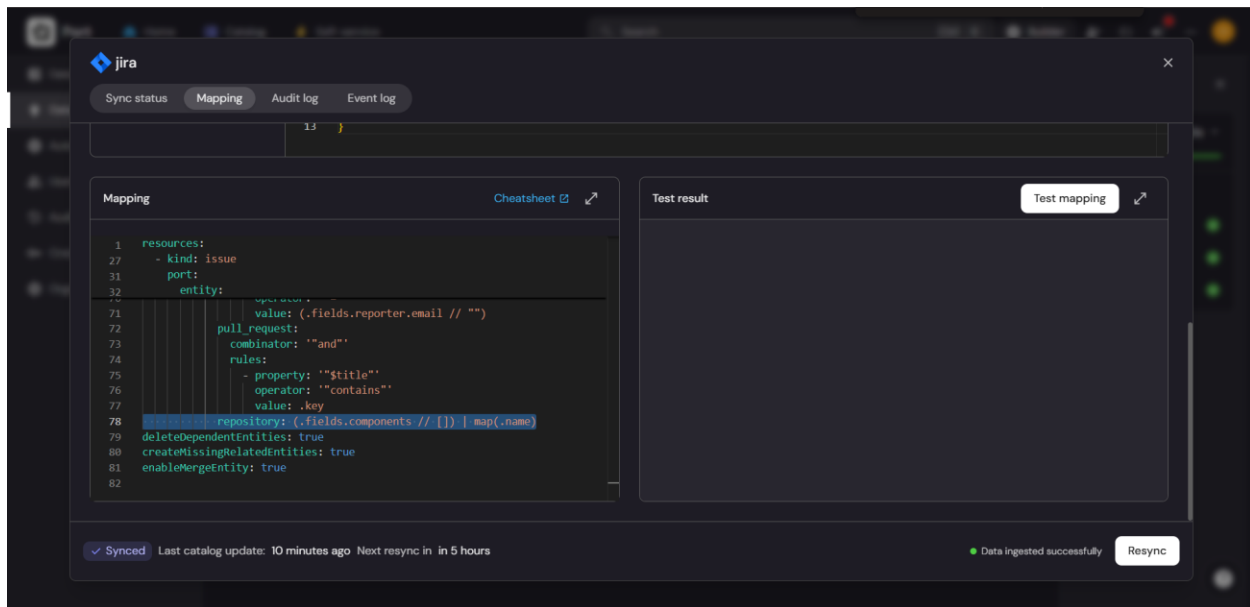
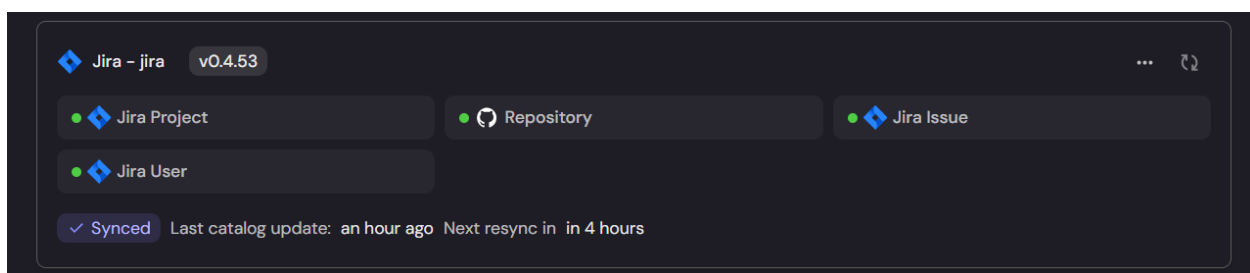
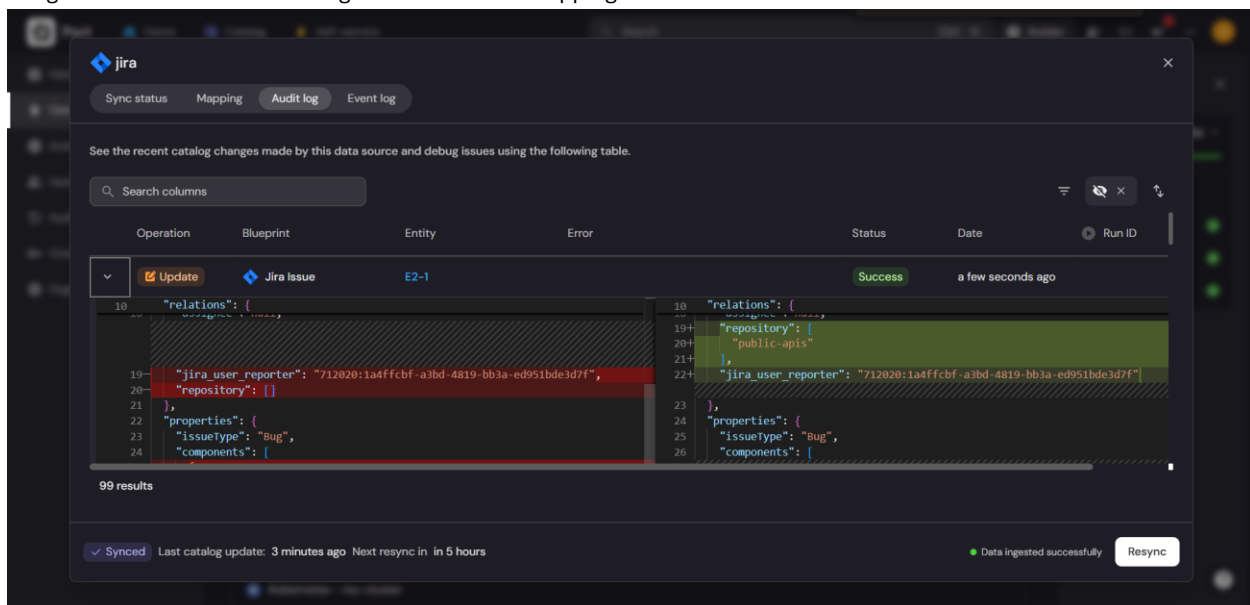


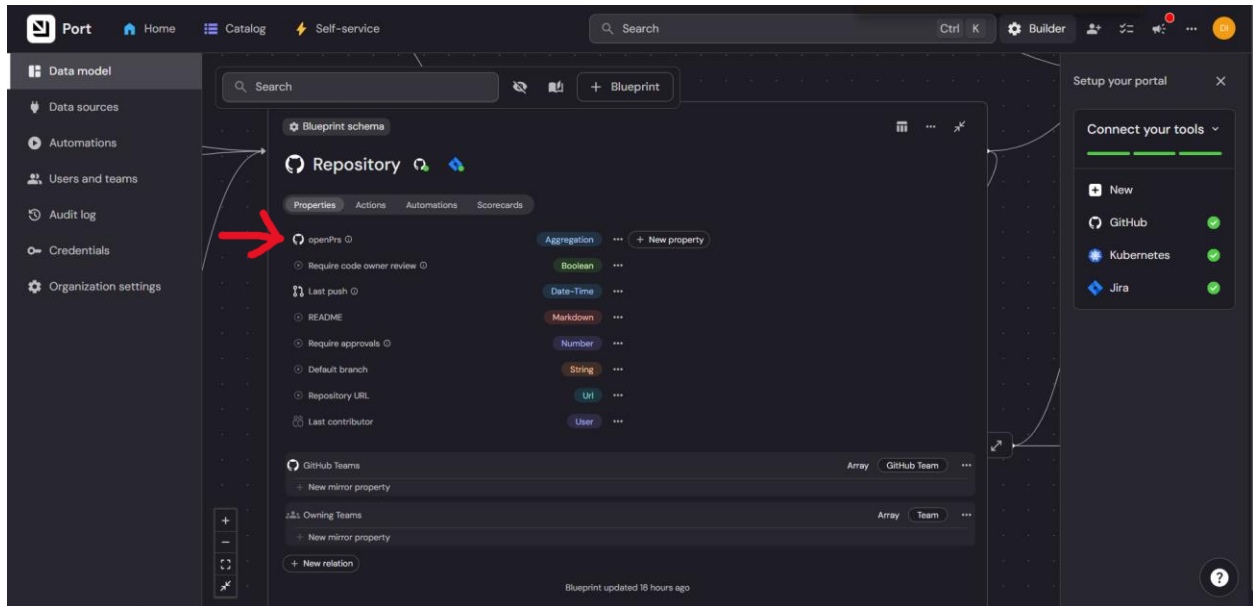
Image below shows the Audit Log to show that the mapping works



Exercise #3 – PR Count Scorecard

1. Added the aggregation property named openPrs

Image below shows the added Aggregation property



Images below show the steps for the Aggregation property setup

Edit property

Icon

Related blueprint* ⓘ

Pull Request

×

▼

Calculate by*

Entities

×

▼

Function*

count

×

▼

Query ⓘ

≡ 1 filter

Path Filter ⓘ

Choose a value

▼

Cancel

Save

Filters

Edit JSON

X

Where

Status

=

open

X

+ Filter

Save

2. Added the scorecard logic to the Repository blueprint

...

```
{
  "identifier": "open_prs_scorecard",
  "title": "Open PRs per Repository",
  "levels": [
    {
      "color": "paleBlue",
      "title": "Basic"
    },
    {
      "color": "bronze",
      "title": "Bronze"
    },
    {
      "color": "silver",
      "title": "Silver"
    },
    {
      "color": "gold",
      "title": "Gold"
    }
  ],
  "rules": [
    {
      "identifier": "open_prs_gold",
```

```
"title": "Gold: < 5 Open PRs",
"level": "Gold",
"query": {
  "combinator": "and",
  "conditions": [
    {
      "operator": "<",
      "property": "open_prs",
      "value": 5
    }
  ]
},
{
  "identifier": "open_prs_silver",
  "title": "Silver: < 10 Open PRs",
  "level": "Silver",
  "query": {
    "combinator": "and",
    "conditions": [
      {
        "operator": "<",
        "property": "open_prs",
        "value": 10
      }
    ]
  }
},
{
  "identifier": "open_prs_bronze",
  "title": "Bronze: < 15 Open PRs",
  "level": "Bronze",
  "query": {
    "combinator": "and",
    "conditions": [
      {
        "operator": "<",
        "property": "open_prs",
        "value": 15
      }
    ]
  }
}
]
```

```
}  
` ` `
```

Image below shows the scorecard for the repository EagleEye with 1 open PR

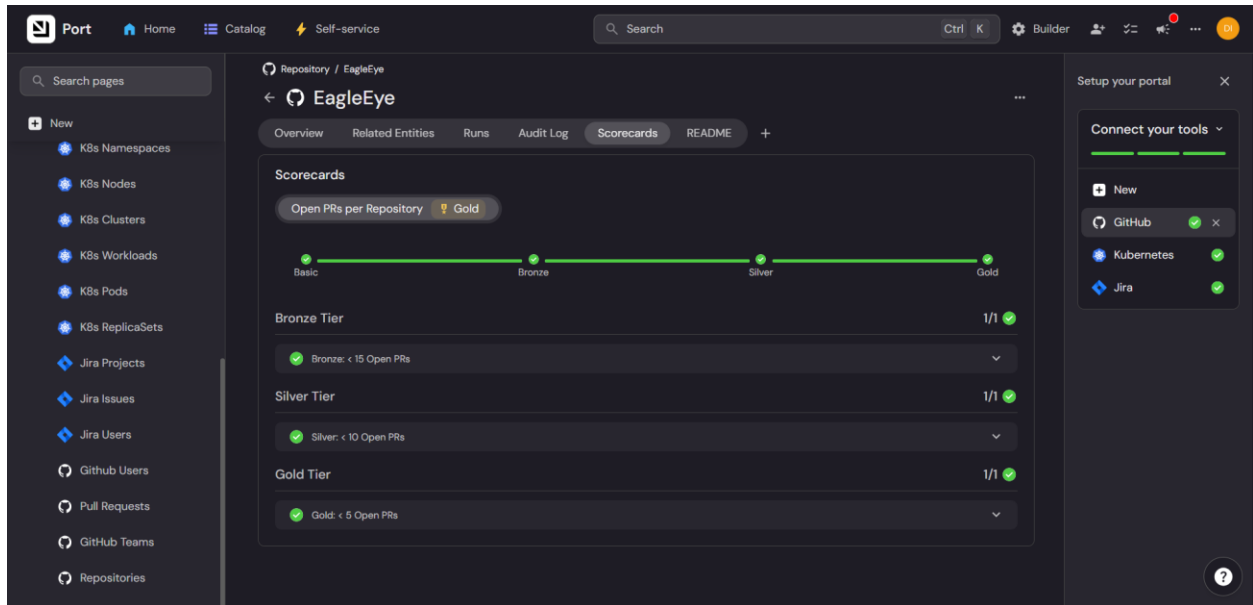
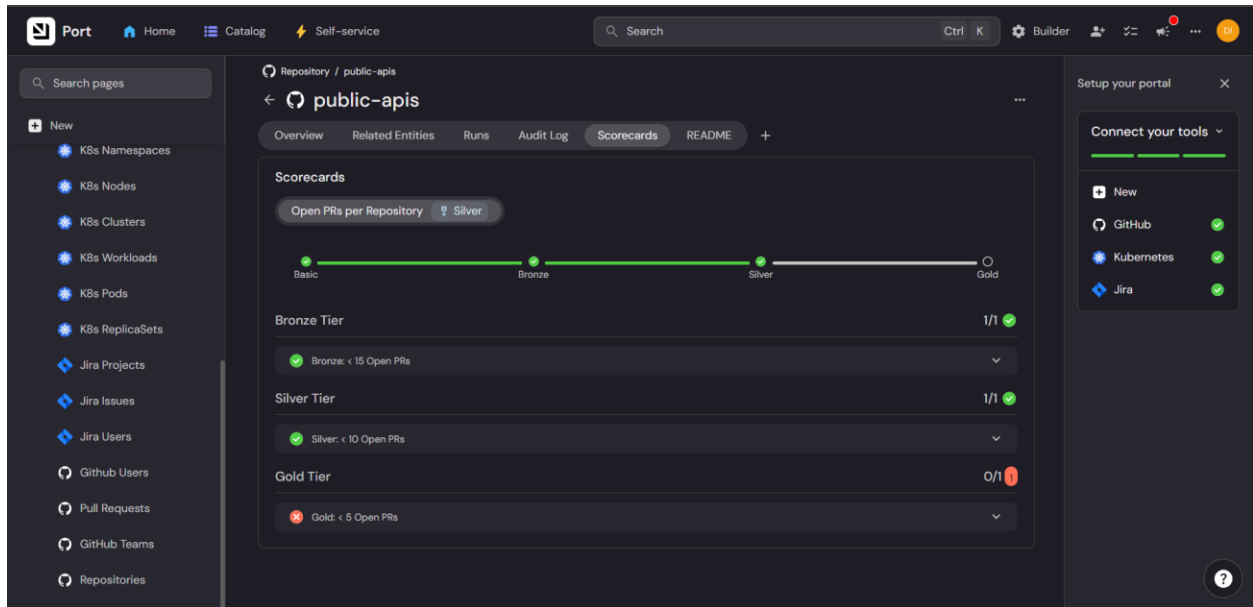


Image below shows the scorecard for the repository public-apis with 6 open PR



Port

Home

Catalog

Self-service

Search

Ctrl K

Builder

Search pages

New

K8s Namespaces

K8s Nodes

K8s Clusters

K8s Workloads

K8s Pods

K8s ReplicaSets

Jira Projects

Jira Issues

Jira Users

Github Users

Pull Requests

Github Teams

Repositories

Repositories

Search columns

vals

openPrs

Open PRs per Repository

Gold: < 5 Open PRs

Silver: < 10 Open PRs

Bronze: < 15 Open PRs

Property

0	Gold	Passed	Passed	Passed	...
0	Gold	Passed	Passed	Passed	...
1	Gold	Passed	Passed	Passed	...
1	Gold	Passed	Passed	Passed	...
6	Silver	Not passed	Passed	Passed	...
0	Gold	Passed	Passed	Passed	...
0	Gold	Passed	Passed	Passed	...
<div><div></div></div>		89%	100%	100%	

9 results

Setup your portal

Connect your tools

New

Github

Kubernetes

Jira

Exercise #4 – GitHub Workflow Troubleshooting.

Step 1. Verify Action Backend Configuration

Organization, Repository, and Workflow Name:

Double-check that the self-service action in Port is configured with the correct GitHub organization, repository, and workflow file name. Any typo or mismatch will most likely prevent the workflow from being triggered. This is a common cause and will result in the action being stuck in progress with no activity in GitHub.

Workflow Location:

Ensure the workflow file exists in the `.github/workflows/` directory of the specified repository and branch.

Step 2. GitHub App Installation**App Installation:**

Confirm that Port's GitHub App is installed in the correct GitHub organization and on the relevant repository.

Permissions:

The GitHub App must have the necessary permissions (Actions, Pull Requests, Contents, etc.) and be installed on the repository where the workflow resides.

Step 3. Workflow Trigger Configuration**workflow_dispatch Trigger:**

The workflow must be configured to use the `workflow_dispatch` trigger. Without this, it cannot be triggered remotely by Port.

Branch Existence:

If specifying a branch via the `ref` key, customer has to ensure the workflow file exists in the default branch as well, due to GitHub's requirements.

Step 4. Secrets and Authentication**Port Credentials:**

Ensure the GitHub repository contains the correct `PORT_CLIENT_ID` and `PORT_CLIENT_SECRET` as secrets, if required by your workflow.

GitHub Token:

If using a webhook invocation, verify that the GitHub token used has the necessary permissions and is correctly referenced in Port's secrets.

Step 5. Logs and Error Messages

Check Port UI and Logs:

Look for any error messages in the Port UI or logs. Sometimes, errors such as “Got 404 while addressing github api, repo or workflow not found” indicate misconfiguration of org, repo, or workflow names.

GitHub Workflow Runs:

Check the Actions tab in the GitHub repository to see if any workflow runs were triggered or if there are any failed runs.

Step 6. Additional Checks

Action Organization/Repository:

If using templates or guides, ensure you have replaced all placeholder values (e.g., <GITHUB_ORG>, <GITHUB_REPO>) with your actual organization and repository names.

Secrets in Port:

Make sure all required secrets are set in Port (e.g., GitHub tokens, Port credentials).