Sure! Let’s walk through the step-by-step process of setting up and deploying two Spring Boot microservices using Azure Repos for version control, Azure Pipelines for CI/CD, and OpenShift for deployment and runtime management. I’ll also include real-time use case configurations.

**1. Development and Version Control**

**Step 1: Code Setup in Azure Repos**

* **Create a Repository:** Start by creating a repository in Azure Repos. Let’s say you have two microservices, user-service and order-service.
* **Directory Structure:**
  + You might organize your repository as follows:

css

Copy code

/azure-repo

├── user-service

│ ├── src

│ └── Dockerfile

└── order-service

├── src

└── Dockerfile

**2. Continuous Integration (CI) with Azure Pipelines**

**Step 2: Create Azure Pipeline for CI**

* **Define Pipeline (YAML):** Create a azure-pipelines.yml file at the root of your repository.
* **Sample CI Pipeline Configuration:**

yaml

Copy code

trigger:

branches:

include:

- main

pool:

vmImage: 'ubuntu-latest'

jobs:

- job: Build\_and\_Publish

steps:

- task: Maven@3

inputs:

mavenPomFile: '$(Build.SourcesDirectory)/user-service/pom.xml'

goals: 'clean package'

- task: Maven@3

inputs:

mavenPomFile: '$(Build.SourcesDirectory)/order-service/pom.xml'

goals: 'clean package'

- task: Docker@2

displayName: 'Build and Push Docker Image - user-service'

inputs:

containerRegistry: '$(dockerRegistryServiceConnection)'

repository: 'myregistry/user-service'

dockerfile: '$(Build.SourcesDirectory)/user-service/Dockerfile'

tags: |

$(Build.BuildId)

- task: Docker@2

displayName: 'Build and Push Docker Image - order-service'

inputs:

containerRegistry: '$(dockerRegistryServiceConnection)'

repository: 'myregistry/order-service'

dockerfile: '$(Build.SourcesDirectory)/order-service/Dockerfile'

tags: |

$(Build.BuildId)

* **Explanation:**
  + **Trigger:** Pipeline triggers on commits to the main branch.
  + **Build:** The pipeline runs Maven commands to build both microservices.
  + **Docker Build and Push:** It then builds Docker images for each microservice and pushes them to an Azure Container Registry (ACR).

**3. Continuous Deployment (CD) with Azure Pipelines**

**Step 3: Create Azure Pipeline for CD**

* **Define CD Pipeline Configuration:**

yaml

Copy code

trigger: none

resources:

containers:

- container: user-service-image

type: ACR

azureSubscription: 'your-azure-subscription'

resourceGroup: 'your-resource-group'

registry: 'your-acr-name'

repository: 'user-service'

tag: 'latest'

- container: order-service-image

type: ACR

azureSubscription: 'your-azure-subscription'

resourceGroup: 'your-resource-group'

registry: 'your-acr-name'

repository: 'order-service'

tag: 'latest'

stages:

- stage: Deploy

jobs:

- deployment: DeployToOpenShift

environment: 'OpenShift'

strategy:

runOnce:

deploy:

steps:

- task: Kubernetes@1

displayName: 'Deploy user-service to OpenShift'

inputs:

connectionType: 'Kubernetes Service Connection'

kubernetesServiceEndpoint: 'your-k8s-service-connection'

namespace: 'your-namespace'

command: 'apply'

arguments: '-f $(Pipeline.Workspace)/manifests/user-service-deployment.yaml'

- task: Kubernetes@1

displayName: 'Deploy order-service to OpenShift'

inputs:

connectionType: 'Kubernetes Service Connection'

kubernetesServiceEndpoint: 'your-k8s-service-connection'

namespace: 'your-namespace'

command: 'apply'

arguments: '-f $(Pipeline.Workspace)/manifests/order-service-deployment.yaml'

* **Explanation:**
  + **Resources:** Specifies the container images to deploy from ACR.
  + **Deploy Stage:** Deploys the microservices to OpenShift using Kubernetes manifests.

**4. Configurations and Deployment in OpenShift**

**Step 4: Prepare Kubernetes Manifests**

* **Create Deployment Manifests:**
  + **user-service-deployment.yaml:**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: user-service

spec:

replicas: 2

selector:

matchLabels:

app: user-service

template:

metadata:

labels:

app: user-service

spec:

containers:

- name: user-service

image: myregistry/user-service:latest

ports:

- containerPort: 8080

* + **order-service-deployment.yaml:**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: order-service

spec:

replicas: 2

selector:

matchLabels:

app: order-service

template:

metadata:

labels:

app: order-service

spec:

containers:

- name: order-service

image: myregistry/order-service:latest

ports:

- containerPort: 8081

* **Service and Route Configurations:** You may also need to create Kubernetes Service and Route objects to expose your microservices.

**Step 5: Deploy Microservices to OpenShift**

* **Use the Azure Pipeline CD:** The CD pipeline will automatically apply the Kubernetes manifests to OpenShift, deploying your microservices.
* **Logging and Monitoring:**
  + Access logs via OpenShift’s console or CLI:

bash

Copy code

oc logs -f deployment/user-service

oc logs -f deployment/order-service

* + OpenShift provides integrated monitoring tools to view metrics and monitor the health of your microservices.

**5. Real-Time Use Case Example: User and Order Management**

* **user-service:** Manages user data, such as user registration, login, and profile management.
* **order-service:** Handles order processing, including creating orders, updating order status, and fetching order details.

**6. Summary of Configuration Details**

* **Azure Repos:** Source code repository.
* **Azure Pipelines:** For CI/CD, including building, testing, containerizing, and deploying microservices.
* **Azure Container Registry (ACR):** Stores Docker images of microservices.
* **OpenShift:** Runs the microservices in a Kubernetes environment, managing scaling, routing, and monitoring.
* **Namespace:** Both microservices are deployed in the same namespace, enabling them to communicate internally.

This setup ensures a seamless process from code development to deployment and monitoring in a real-time production environment using Azure and OpenShift.