By Chahine Jebabli & Khalil Elachkham

IT460: Project Microservices Deployment Report

Introduction

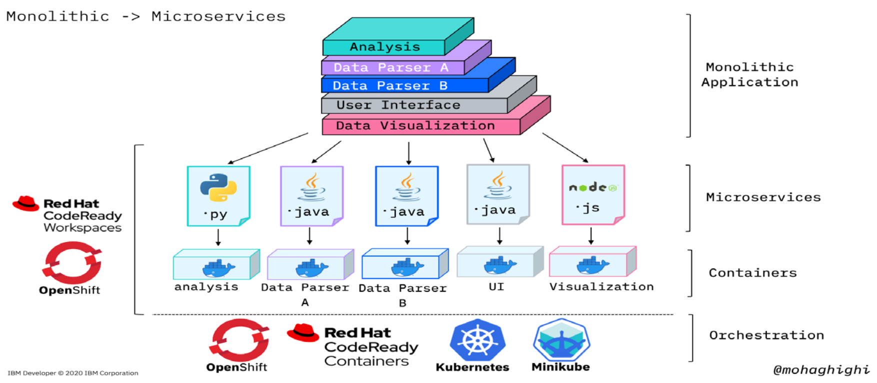
This report presents a comprehensive overview of the deployment process and architecture for various microservices within our project. Our project integrates a suite of microservices, each tailored for distinct functionalities and interactions within the larger system. The primary focus of this report is on the deployment procedures and the crucial architectural decisions implemented for each microservice.

I- Concepts to learn

1- Microservices:

- ❖ Microservices architecture advocates partitioning large monolithic applications into smaller independent services that communicate with each other by using HTTP and messages.

- ❖ Services must be:
  - o Loosely coupled.
  - o Independently deployable.
  - o Highly maintainable and testable.

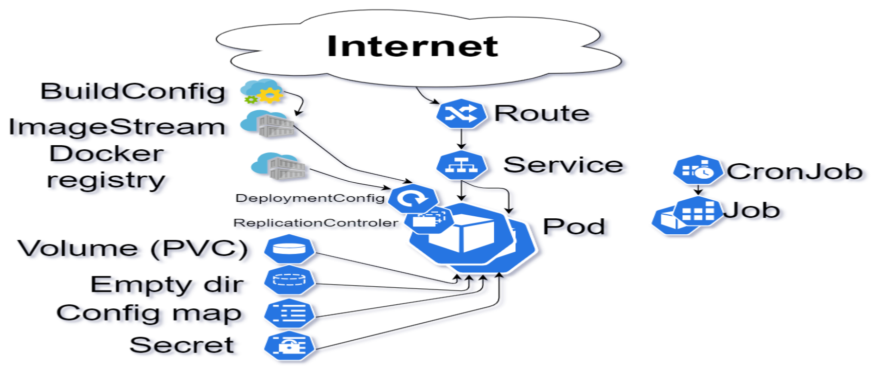


2- What is a container:

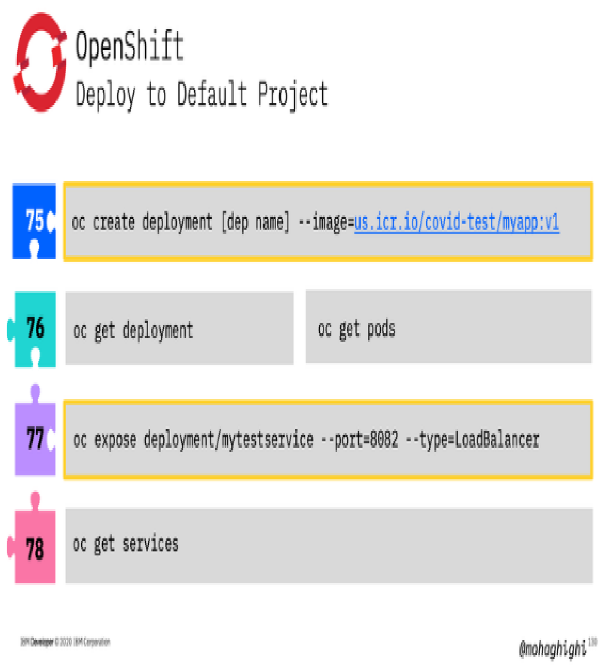
- Containers are executable units of software.
- Application code is packaged into the container.
  - ➢ Including libraries and dependencies.
- Using containers we can run our applications anywhere.
- We use Docker Files to build Docker Images, then run Docker Containers.

3- OpenShift Concepts:

1. **Pods:** Basic units running your app's containers.
2. **Routes:** Makes your app accessible from outside OpenShift.
3. **Deployment:** Handles app lifecycle, like scaling and updates.
4. **DeploymentConfig:** Configurations for deploying apps.
5. **ImageStream:** Central storage for managing Docker images.
6. **Builds:** Creating Docker images from source code.
7. **Persistent Volume:** Storage that keeps data even if pods restart.
8. **YAML:** A human-readable data serialization format used in OpenShift to define configurations for various resources.



4- OpenShift CLI Commands:



OpenShift
Deploy to Default Project

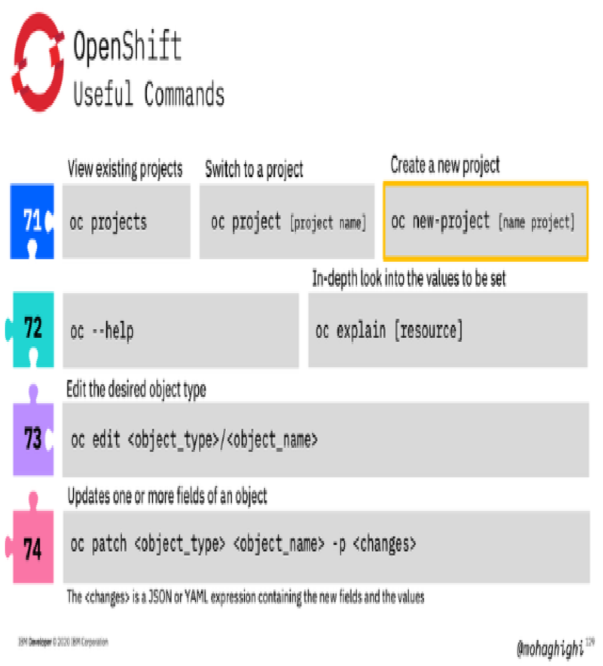
75
```
oc create deployment [dep name] --image=us.icr.io/covid-test/myapp:v1
```

76
```
oc get deployment
```
```
oc get pods
```

77
```
oc expose deployment/mytestservice --port=8082 --type=LoadBalancer
```

78
```
oc get services
```

IBM Developer © 2020 IBM Corporation @mohaghighi 130



OpenShift
Useful Commands

71

View existing projects
```
oc projects
```
Switch to a project
```
oc project [project name]
```
Create a new project
```
oc new-project [name project]
```

72
```
oc --help
```
In-depth look into the values to be set
```
oc explain [resource]
```

73

Edit the desired object type
```
oc edit <object_type>/<object_name>
```

74

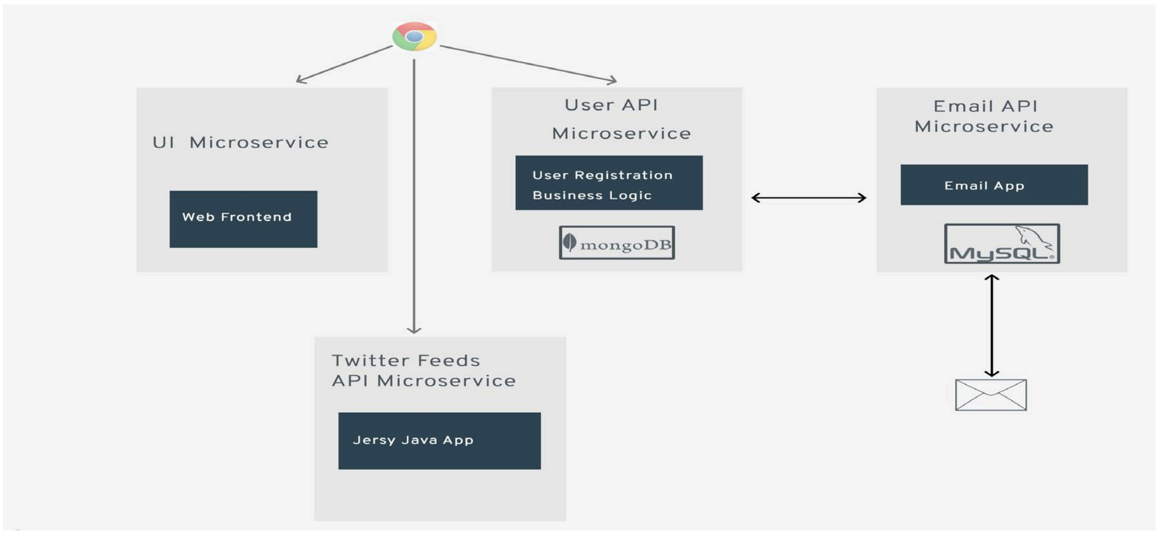
Updates one or more fields of an object
```
oc patch <object_type> <object_name> -p <changes>
```
The <changes> is a JSON or YAML expression containing the new fields and the values

IBM Developer © 2020 IBM Corporation @mohaghighi 129

Project Architecture Overview

Our project comprises a robust architecture designed to deliver seamless functionality through three core microservices. Illustrated in the accompanying diagram, these microservices collectively form the backbone of our application.

1. **UserRegistration Backend:**
    - **F**oundation for user management, offering REST APIs for registration and listing.
    - Developed in NodeJS, ensuring efficient request handling and data persistence in MongoDB.
2. **UserRegistration Frontend:**
    - Frontend microservice in PHP, focused on user-friendly web interface.
    - Crafts visually appealing pages to enhance user experience.
3. **Email Service:**
    - Handles email requests and dispatches emails promptly.
    - Developed in Python, with MySQL integration for email log.
4. **Twitter Feeds API Microservice:**
    - Java application for retrieving recent tweets based on specified username.
    - Seamlessly integrates with Twitter's API for up-to-date information.



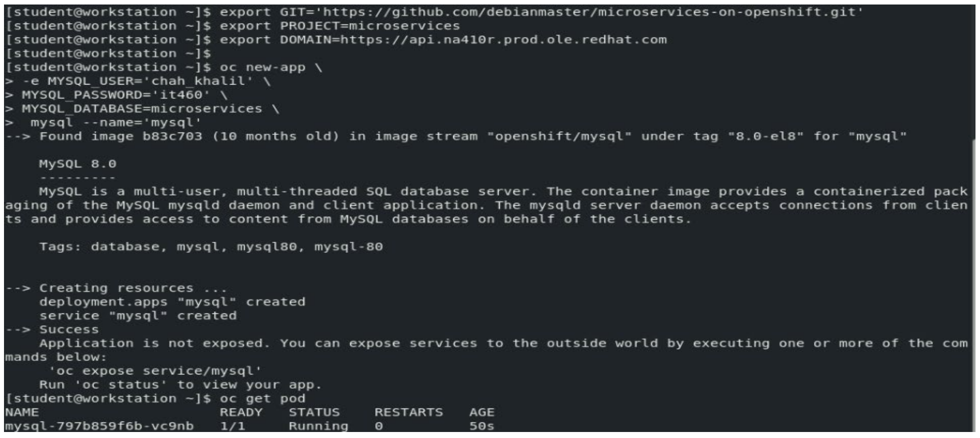
II- Microservice 1: Email Microservice

❖ **Architecture Overview**

- **Type**: Python-based RESTful API
- **Functionality**: Manages email operations, encompassing sending emails and logging information to a MySQL database.
- **Language & Framework**: Python, chosen for its straightforward nature and extensive libraries for email handling.
- **Database Selection**: MySQL, selected for its dependability in storing email logs.
- **Deployment**: Executed as a single pod within OpenShift to ensure isolation.
- **Image**: Incorporates the official OpenShift MySQL image.
- **Persistence**: Employs a **persistent volume** claim (PVC) for data durability.
- **Networking**: Configures the MySQL database for internal exposure within the OpenShift cluster.

❖ **Deployment Process**

- MySQL Database Setup
    - **Deployment**: MySQL deployed as an individual pod.
    - **Image**: Utilizes the official OpenShift MySQL image.(openshift/mysql:8.0-el8)
    - **Persistence**: Implements PVC for data storage.
    - **Networking**: MySQL is internally accessible within the cluster.

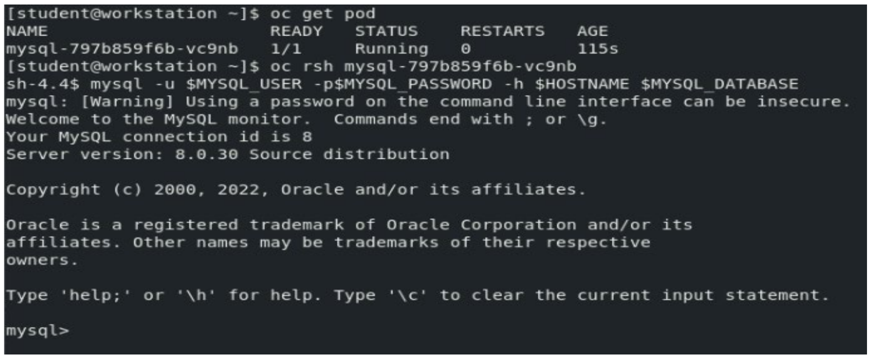


```
[student@workstation ~]$ export GIT='https://github.com/debianmaster/microservices-on-openshift.git'
[student@workstation ~]$ export PROJECT=microservices
[student@workstation ~]$ export DOMAIN=https://api.na410r.prod.ole.redhat.com
[student@workstation ~]$
[student@workstation ~]$ oc new-app \
> -e MYSQL_USER='chah_khalil' \
> MYSQL_PASSWORD='it460' \
> MYSQL_DATABASE=microservices \
>  mysql --name='mysql'
--> Found image b83c703 (10 months old) in image stream "openshift/mysql" under tag "8.0-el8" for "mysql"

    MySQL 8.0
    ---------
    MySQL is a multi-user, multi-threaded SQL database server. The container image provides a containerized pack
aging of the MySQL mysqld daemon and client application. The mysqld server daemon accepts connections from clien
ts and provides access to content from MySQL databases on behalf of the clients.

    Tags: database, mysql, mysql80, mysql-80


--> Creating resources ...
    deployment.apps "mysql" created
    service "mysql" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing one or more of the com
mands below:
     'oc expose service/mysql'
    Run 'oc status' to view your app.
[student@workstation ~]$ oc get pod
NAME                     READY   STATUS    RESTARTS   AGE
mysql-797b859f6b-vc9nb   1/1     Running   0          50s
```

Access MySQL for Troubleshooting **(Communication between Containers)**

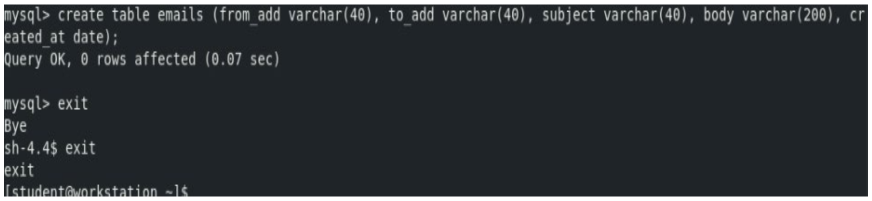
1. **Pod Status**: Verify MySQL pod status with **oc get pod**.
2. **Remote Access**: Initiate a remote shell session in the MySQL pod using **oc rsh <pod-name>**.
3. **Database Server Access**: Utilize the **mysql** command for direct access to the database server.

   ○ MySQL Connection:



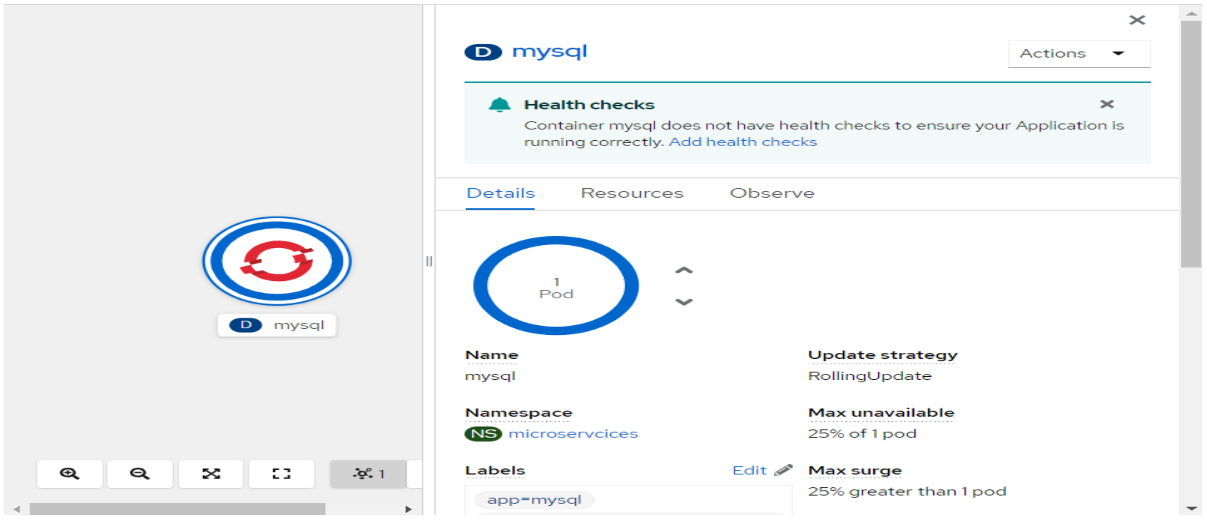
```
[student@workstation ~]$ oc get pod
NAME                     READY   STATUS    RESTARTS   AGE
mysql-797b859f6b-vc9nb   1/1     Running   0          115s
[student@workstation ~]$ oc rsh mysql-797b859f6b-vc9nb
sh-4.4$ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -h $HOSTNAME $MYSQL_DATABASE
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.30 Source distribution

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

   ○ MySQL Table Creation:



```
mysql> create table emails (from_add varchar(40), to_add varchar(40), subject varchar(40), body varchar(200), cr
eated_at date);
Query OK, 0 rows affected (0.07 sec)

mysql> exit
Bye
sh-4.4$ exit
exit
[student@workstation ~]$
```

✓ **MySQL Deployed:**

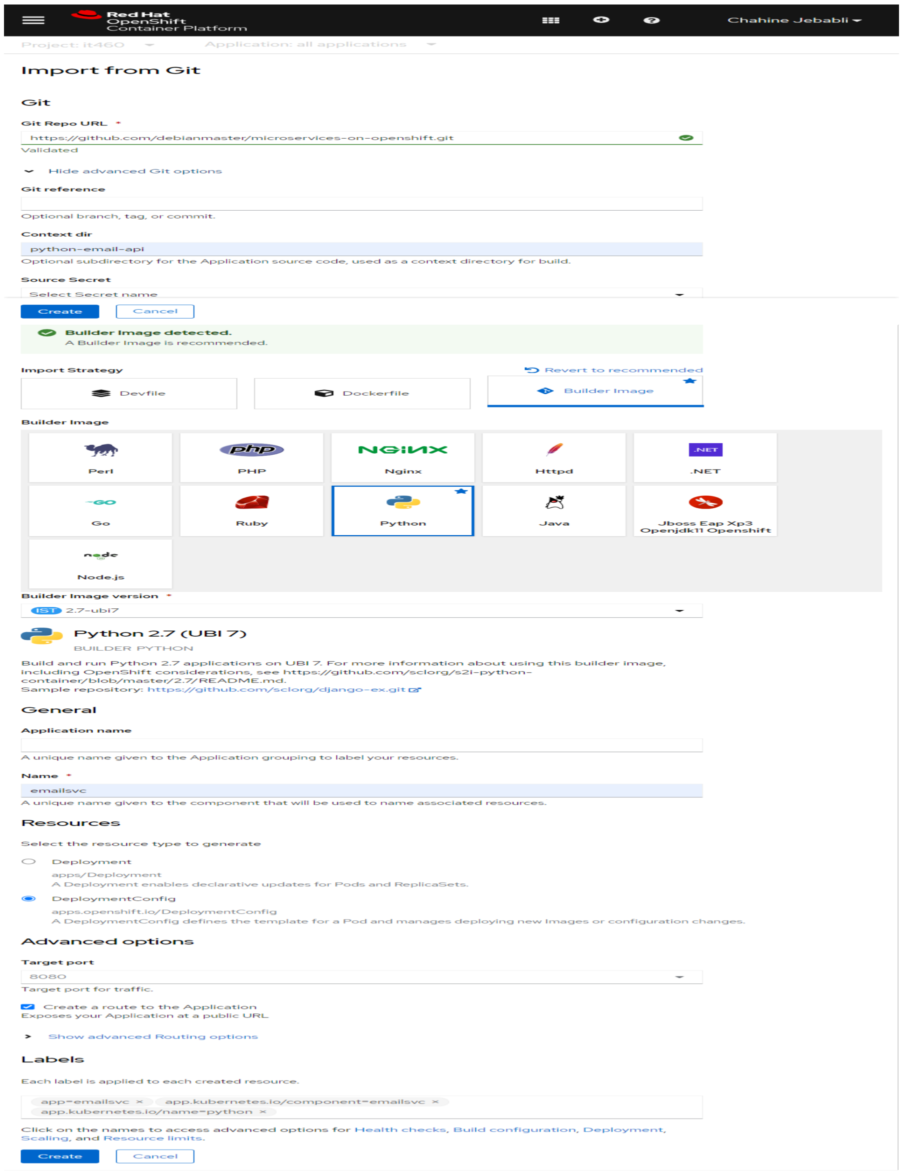


Health checks
Container mysql does not have health checks to ensure your Application is running correctly. Add health checks

Name: mysql
Namespace: NS microservcices
Labels: app=mysql
Update strategy: RollingUpdate
Max unavailable: 25% of 1 pod
Max surge: 25% greater than 1 pod

❖ Email Service Deployment

- **Source Repository Deployment**: Deployed from a specified context directory within the source repository.
- **Configuration**: Sets environment variables for database connectivity and application domain.
- **OpenShift Integration**: Leverages the OpenShift Python 2.7 image stream for deployment.



Import from Git

Git

Git Repo URL *
https://github.com/debianmaster/microservices-on-openshift.git
Validated

Hide advanced Git options

Git reference

Optional branch, tag, or commit.

Context dir
python-email-api
Optional subdirectory for the Application source code, used as a context directory for build.

Source Secret
Select Secret name

Create Cancel

Builder Image detected.
A Builder Image is recommended.

Import Strategy
Revert to recommended
Devfile Dockerfile [✓ Builder Image]

Builder Image
Perl PHP Nginx Httpd .NET Go Ruby [✓ Python] Java Jboss Eap Xp3 Openjdk11 Openshift Node.js

Builder Image version *
IST 2.7-ubi7

Python 2.7 (UBI 7)
BUILDER PYTHON
Build and run Python 2.7 applications on UBI 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-python-container/blob/master/2.7/README.md.
Sample repository: https://github.com/sclorg/django-ex.git

General

Application name

A unique name given to the Application grouping to label your resources.

Name *
emailsvc
A unique name given to the component that will be used to name associated resources.

Resources

Select the resource type to generate

○ Deployment
apps/Deployment
A Deployment enables declarative updates for Pods and ReplicaSets.

● DeploymentConfig
apps.openshift.io/DeploymentConfig
A DeploymentConfig defines the template for a Pod and manages deploying new Images or configuration changes.

Advanced options

Target port
8080
Target port for traffic.

✓ Create a route to the Application
Exposes your Application at a public URL

Show advanced Routing options

Labels

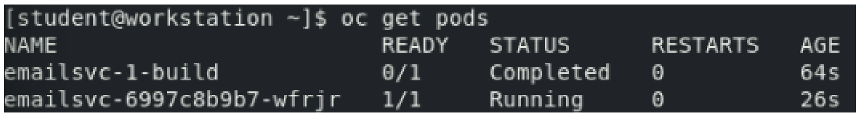
Each label is applied to each created resource.

app=emailsvc × app.kubernetes.io/component=emailsvc × app.kubernetes.io/name=python ×

Click on the names to access advanced options for Health checks, Build configuration, Deployment, Scaling, and Resource limits.

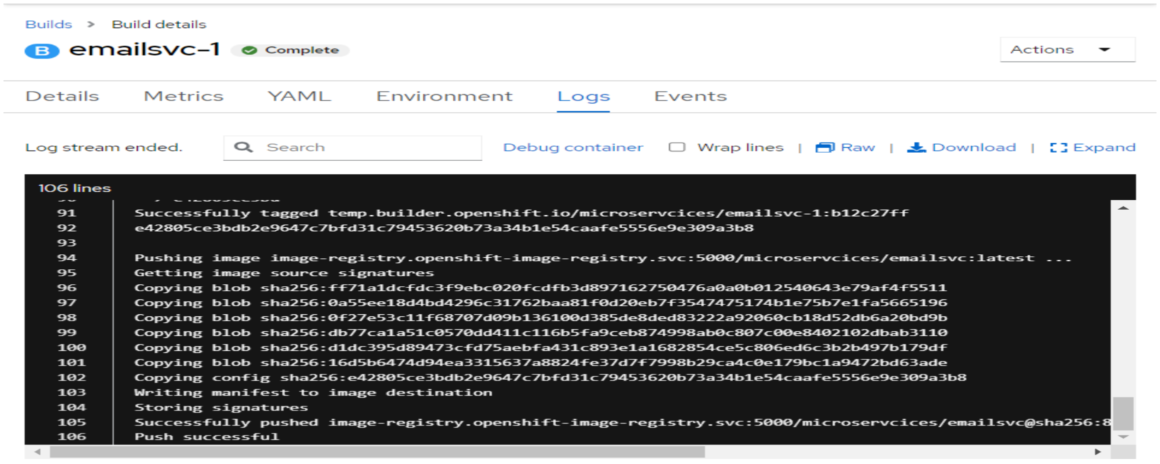
Create Cancel

- **Eamilsvc pods:**



```
[student@workstation ~]$ oc get pods
NAME                         READY   STATUS      RESTARTS   AGE
emailsvc-1-build             0/1     Completed   0          64s
emailsvc-6997c8b9b7-wfrjr    1/1     Running     0          26s
```

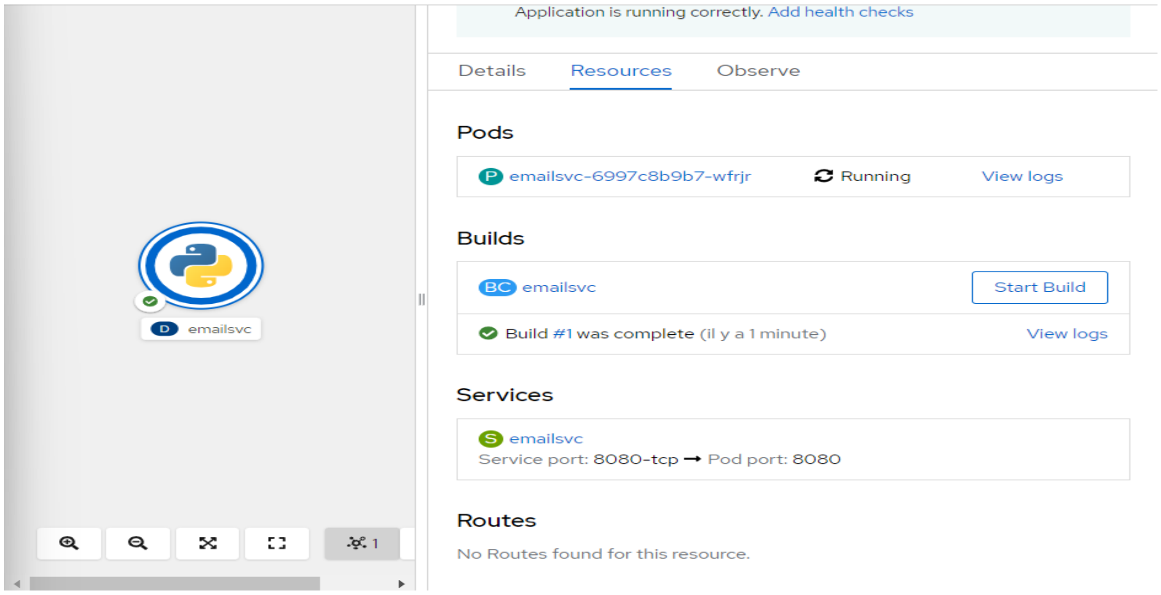
- **Eamilsvc Build Logs:**



Builds > Build details
B emailsvc-1 Complete
Actions
Details Metrics YAML Environment Logs Events
Log stream ended. Search Debug container Wrap lines | Raw | Download | Expand
106 lines

```
Successfully tagged temp.builder.openshift.io/microservcices/emailsvc-1:b12c27ff
e42805ce3bdb2e9647c7bfd31c79453620b73a34b1e54caafe5556e9e309a3b8

Pushing image image-registry.openshift-image-registry.svc:5000/microservcices/emailsvc:latest ...
Getting image source signatures
Copying blob sha256:ff71a1dcfdc3f9ebc020fcdfb3d897162750476a0a0b012540643e79af4f5511
Copying blob sha256:0a55ee18d4bd4296c31762baa81f0d20eb7f3547475174b1e75b7e1fa5665196
Copying blob sha256:0f27e53c11f68707d09b136100d385de8ded83222a92060cb18d52db6a20bd9b
Copying blob sha256:db77ca1a51c0570dd411c116b5fa9ceb874998ab0c807c00e8402102dbab3110
Copying blob sha256:d1dc395d89473cfd75aebfa431c893e1a1682854ce5c806ed6c3b2b497b179df
Copying blob sha256:16d5b6474d94ea3315637a8824fe37d7f7998b29ca4c0e179bc1a9472bd63ade
Copying config sha256:e42805ce3bdb2e9647c7bfd31c79453620b73a34b1e54caafe5556e9e309a3b8
Writing manifest to image destination
Storing signatures
Successfully pushed image-registry.openshift-image-registry.svc:5000/microservcices/emailsvc@sha256:8
Push successful
```

✓ **<u>emailsvc Deployed:</u>**



Application is running correctly. Add health checks

Pods

P emailsvc-6997c8b9b7-wfrjr   Running   View logs

Builds

BC emailsvc   Start Build

Build #1 was complete (il y a 1 minute)   View logs

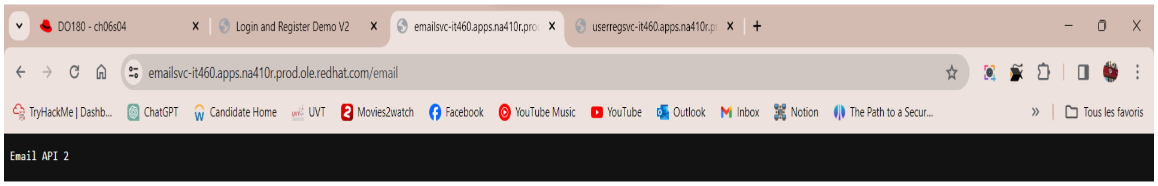
Services

S emailsvc
Service port: 8080-tcp → Pod port: 8080

Routes

No Routes found for this resource.

✓ **<u>emailsvc testing route:</u>**



DO180 - ch06s04
Login and Register Demo V2
emailsvc-it460.apps.na410r.pro
userregsvc-it460.apps.na410r.p
TryHackMe | Dashb... ChatGPT Candidate Home UVT Movies2watch Facebook YouTube Music YouTube Outlook Inbox Notion The Path to a Secur... Tous les favoris

emailsvc-it460.apps.na410r.prod.ole.redhat.com/email

```
Email API 2
```

III- Microservice 2: User Registration Backend Microservice (Node.js Application)

Architecture Overview

- **Type**: Node.js microservice focused on business logic.
- **Functionality**: Offers REST APIs for user registration and listing.
- **Data Persistence**: Engages with MongoDB for storing user data.
- **Language**: Node.js, selected for its non-blocking, event-driven architecture.
- **Database**: MongoDB, preferred for its **scalability** and flexibility.

➢ The User Registration Backend Microservice consists of two key components. Firstly, it employs a MongoDB database to store user data. Secondly, it comprises a business logic layer that provides REST APIs for user registration and user listing functionalities, developed using Node.js.

A. Create a MongoDB Database and Expose it as an Internal Service:

❖ **Deployment Process**

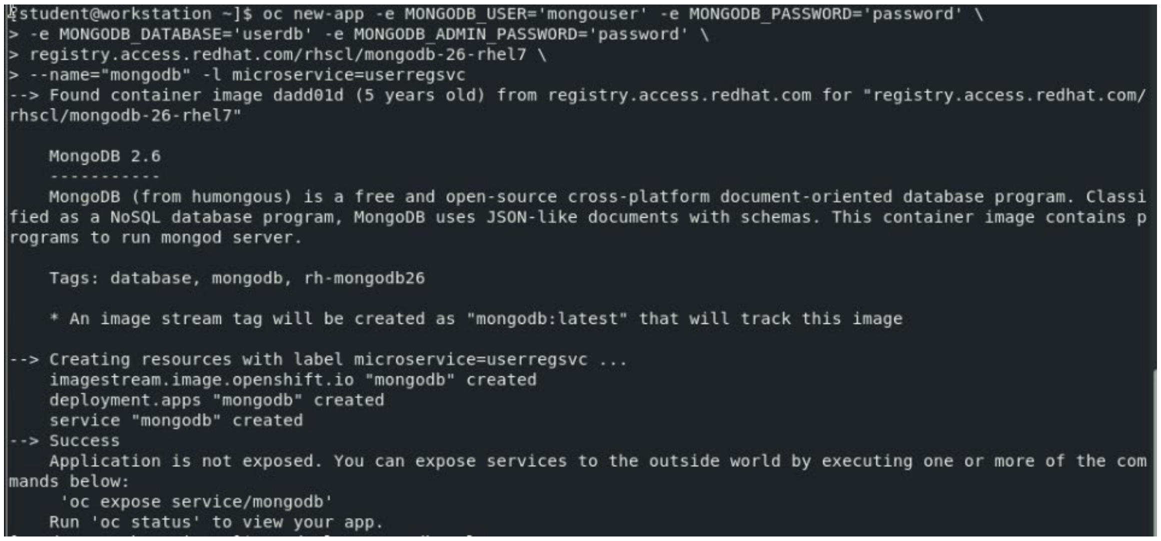
MongoDB Database Creation

1. **Configuration**: Initiate database setup with **oc new-app , set env -e**
2. **Database Deployment**: Deploy the database using **oc deploy mongodb --latest**.

User Registration Service Deployment

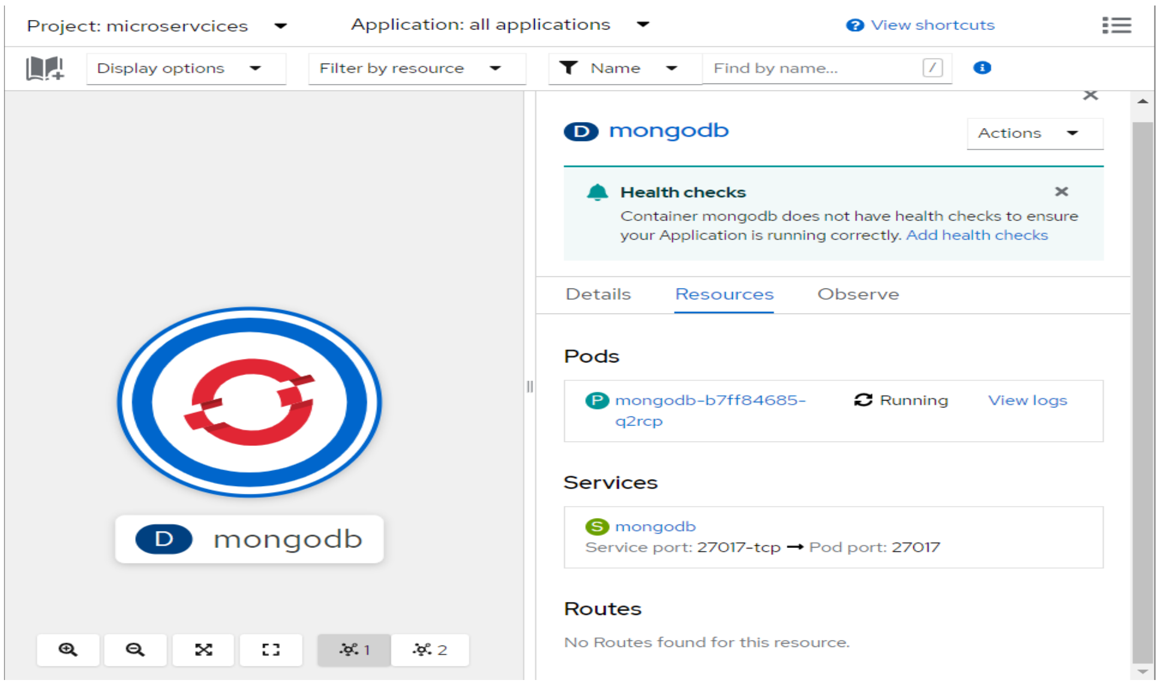
1. **Service Deployment**: Deploy the service from the GitHub repository, specifying the Node.js context directory.
2. **Environment Configuration**: Set environment variables for MongoDB connectivity and Email application domain.
3. **Service Exposure**: Expose the service using **oc expose svc/userregsvc**.

- **Database Deployment**: Deploy MongoDB using **oc new-app**, indicating the image stream and environment variables.
- **Persistence Setup**: Organize PVC for data sustainability.
- **Networking**: Internally expose the MongoDB service within the OpenShift cluster.



```
[student@workstation ~]$ oc new-app -e MONGODB_USER='mongouser' -e MONGODB_PASSWORD='password' \
> -e MONGODB_DATABASE='userdb' -e MONGODB_ADMIN_PASSWORD='password' \
> registry.access.redhat.com/rhscl/mongodb-26-rhel7 \
> --name="mongodb" -l microservice=userregsvc
--> Found container image dadd01d (5 years old) from registry.access.redhat.com for "registry.access.redhat.com/
rhscl/mongodb-26-rhel7"

    MongoDB 2.6
    -----------
    MongoDB (from humongous) is a free and open-source cross-platform document-oriented database program. Classi
fied as a NoSQL database program, MongoDB uses JSON-like documents with schemas. This container image contains p
rograms to run mongod server.

    Tags: database, mongodb, rh-mongodb26

    * An image stream tag will be created as "mongodb:latest" that will track this image

--> Creating resources with label microservice=userregsvc ...
    imagestream.image.openshift.io "mongodb" created
    deployment.apps "mongodb" created
    service "mongodb" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing one or more of the com
mands below:
     'oc expose service/mongodb'
    Run 'oc status' to view your app.
```

✓ **MongoDB Deployed:**



Health checks
Container mongodb does not have health checks to ensure your Application is running correctly. Add health checks

Pods
mongodb-b7ff84685-q2rcp Running View logs

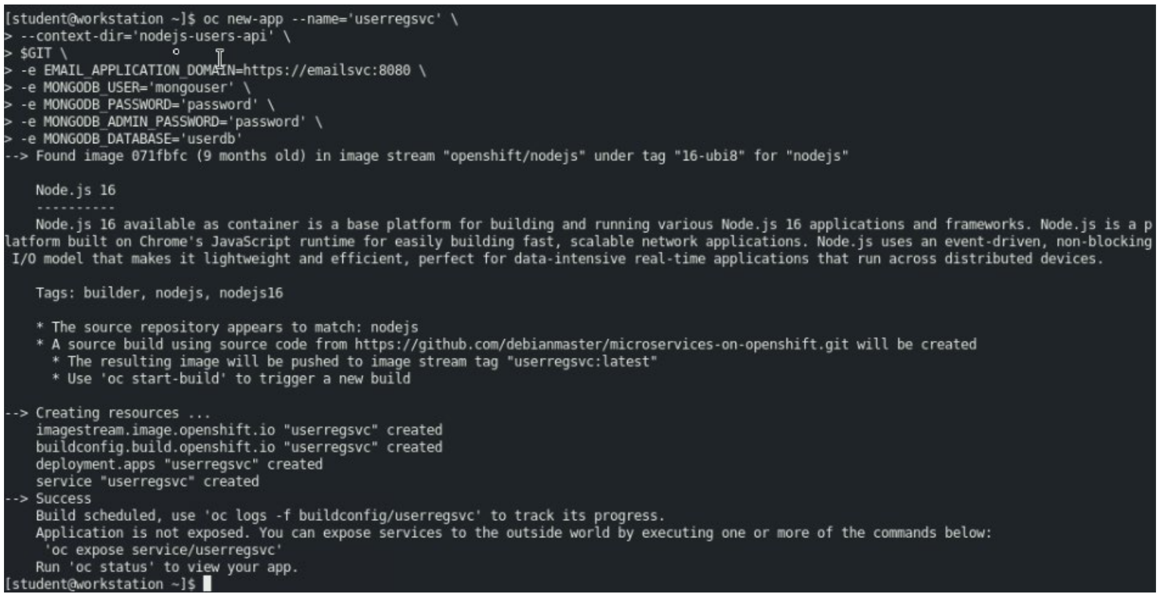
Services
mongodb
Service port: 27017-tcp → Pod port: 27017

Routes
No Routes found for this resource.

Additional Features

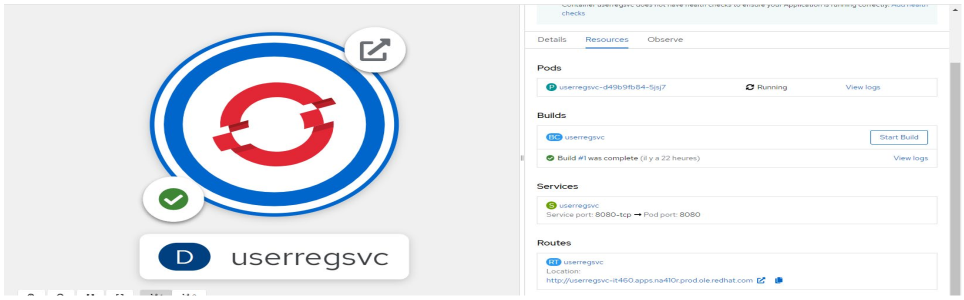
- **Database Interaction**: Executes CRUD operations on user data in MongoDB.
- **Email Microservice Integration**: Collaborates with the Email Microservice for dispatching registration emails.
- **Internal Service Communication**: Utilizes internal service names for efficient intra-cluster communication.

B. <u>Create the User Registration Service and Expose it:</u>



```
[student@workstation ~]$ oc new-app --name='userregsvc' \
> --context-dir='nodejs-users-api' \
> $GIT \
> -e EMAIL_APPLICATION_DOMAIN=https://emailsvc:8080 \
> -e MONGODB_USER='mongouser' \
> -e MONGODB_PASSWORD='password' \
> -e MONGODB_ADMIN_PASSWORD='password' \
> -e MONGODB_DATABASE='userdb'
--> Found image 071fbfc (9 months old) in image stream "openshift/nodejs" under tag "16-ubi8" for "nodejs"

    Node.js 16
    ----------
    Node.js 16 available as container is a base platform for building and running various Node.js 16 applications and frameworks. Node.js is a p
latform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking
 I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

    Tags: builder, nodejs, nodejs16

    * The source repository appears to match: nodejs
    * A source build using source code from https://github.com/debianmaster/microservices-on-openshift.git will be created
      * The resulting image will be pushed to image stream tag "userregsvc:latest"
      * Use 'oc start-build' to trigger a new build

--> Creating resources ...
    imagestream.image.openshift.io "userregsvc" created
    buildconfig.build.openshift.io "userregsvc" created
    deployment.apps "userregsvc" created
    service "userregsvc" created
--> Success
    Build scheduled, use 'oc logs -f buildconfig/userregsvc' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose service/userregsvc'
    Run 'oc status' to view your app.
[student@workstation ~]$
```

✓ **Userregsvc deployed:**



checks

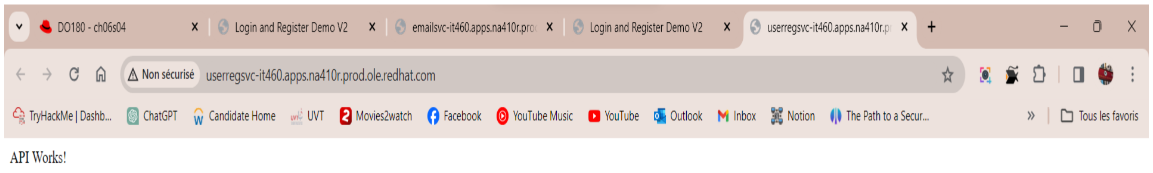
Pods
userregsvc-d49b9fb84-5jsj7 Running

Builds
userregsvc
Build #1 was complete (il y a 22 heures)

Services
userregsvc
Service port: 8080-tcp → Pod port: 8080

Routes
userregsvc
Location:
http://userregsvc-it460.apps.na410r.prod.ole.redhat.com

✓ **Userregsvc API tested:**



userregsvc-it460.apps.na410r.prod.ole.redhat.com

API Works!

IV- Microservice 3: Twitter Feeds API Microservice

Architecture Overview

- **Type**: Java microservice designed for retrieving recent tweets from a specified Twitter username.
- **Functionality**: Accepts a Twitter username as input and returns recent tweets.
- **Image Stream**: Employs the Tomcat 8 image stream from Red Hat JBoss Web Server for deployment.

Deployment Process

Import Tomcat 8 Image (Containerization with OpenShift)

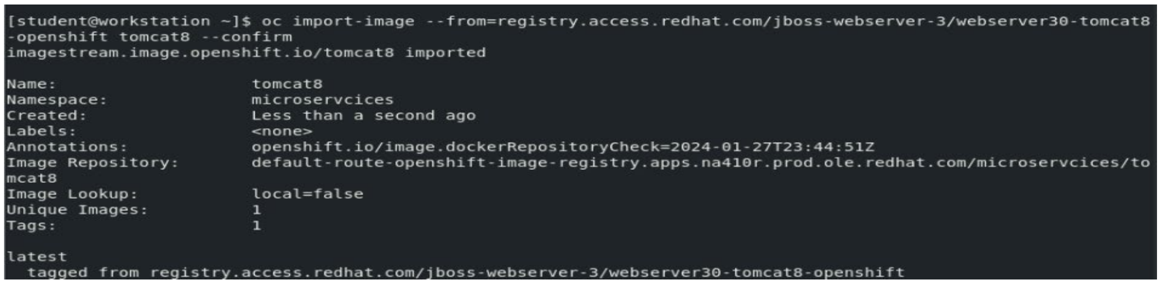
1. **Image Import**: Utilize **oc import-image** to import the Tomcat 8 image from the Red Hat container registry.
2. **Confirmation**: Validate the import to ensure the image is available for deployment.

Deploy the Twitter Feeds API Microservice

1. **Deployment Initialization**: Deploy the microservice from the specified GitHub repository using **oc new-app**.
2. **Repository Context**: Apply the **--context-dir** flag with the path to 'java-twitter-feed-api' in the repository.
3. **Image Selection**: Use the **--image-stream** flag to choose the Tomcat 8 image for deployment.
4. **Naming and Labeling**: Name the microservice 'twitter-api' and label it as **microservice=twittersvc**.

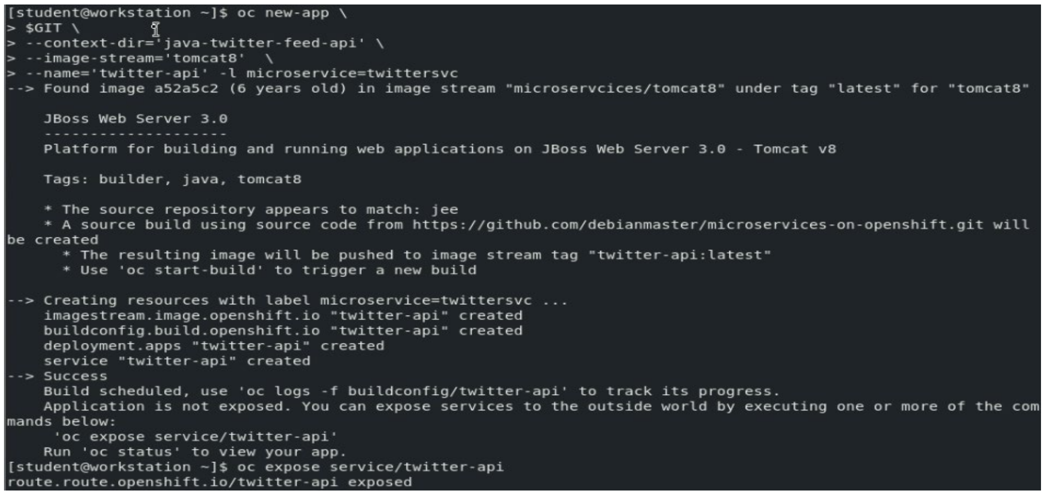
Expose the Service

- **Service Exposure**: Execute **oc expose svc/twitter-api** to make the Twitter Feeds API microservice accessible via a URL
    - **Tomcat8 ImageStream:**



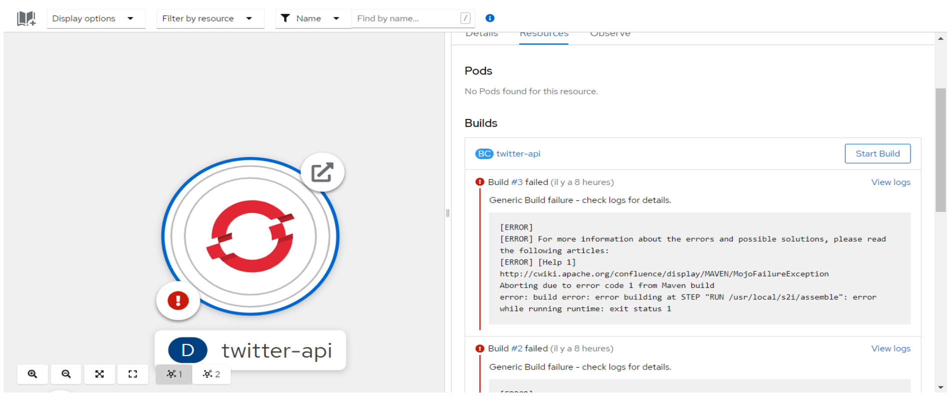
```
[student@workstation ~]$ oc import-image --from=registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat8
-openshift tomcat8 --confirm
imagestream.image.openshift.io/tomcat8 imported

Name:                   tomcat8
Namespace:              microservcices
Created:                Less than a second ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2024-01-27T23:44:51Z
Image Repository:       default-route-openshift-image-registry.apps.na410r.prod.ole.redhat.com/microservcices/to
mcat8
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

latest
  tagged from registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat8-openshift
```

- **Twitter API Build:**



```
[student@workstation ~]$ oc new-app \
> $GIT \
> --context-dir='java-twitter-feed-api' \
> --image-stream='tomcat8'  \
> --name='twitter-api' -l microservice=twittersvc
--> Found image a52a5c2 (6 years old) in image stream "microservcices/tomcat8" under tag "latest" for "tomcat8"

    JBoss Web Server 3.0
    --------------------
    Platform for building and running web applications on JBoss Web Server 3.0 - Tomcat v8

    Tags: builder, java, tomcat8

    * The source repository appears to match: jee
    * A source build using source code from https://github.com/debianmaster/microservices-on-openshift.git will
be created
      * The resulting image will be pushed to image stream tag "twitter-api:latest"
      * Use 'oc start-build' to trigger a new build

--> Creating resources with label microservice=twittersvc ...
    imagestream.image.openshift.io "twitter-api" created
    buildconfig.build.openshift.io "twitter-api" created
    deployment.apps "twitter-api" created
    service "twitter-api" created
--> Success
    Build scheduled, use 'oc logs -f buildconfig/twitter-api' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one or more of the com
mands below:
     'oc expose service/twitter-api'
    Run 'oc status' to view your app.
[student@workstation ~]$ oc expose service/twitter-api
route.route.openshift.io/twitter-api exposed
```

- **CHALLENGE : Build Failed X**



Pods

No Pods found for this resource.

Builds

BC twitter-api — Start Build

Build #3 failed (il y a 8 heures) — View logs

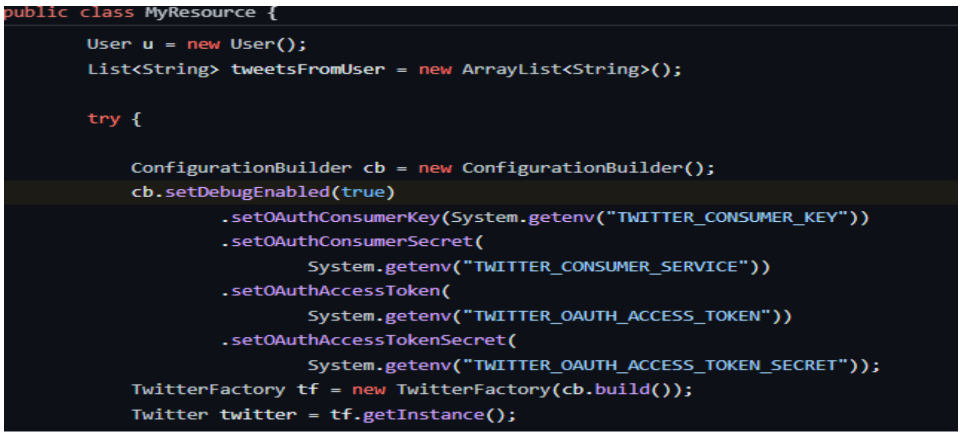
Generic Build failure - check logs for details.

```
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read
the following articles:
[ERROR] [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
Aborting due to error code 1 from Maven build
error: build error: error building at STEP "RUN /usr/local/s2i/assemble": error
while running runtime: exit status 1
```

Build #2 failed (il y a 8 heures) — View logs

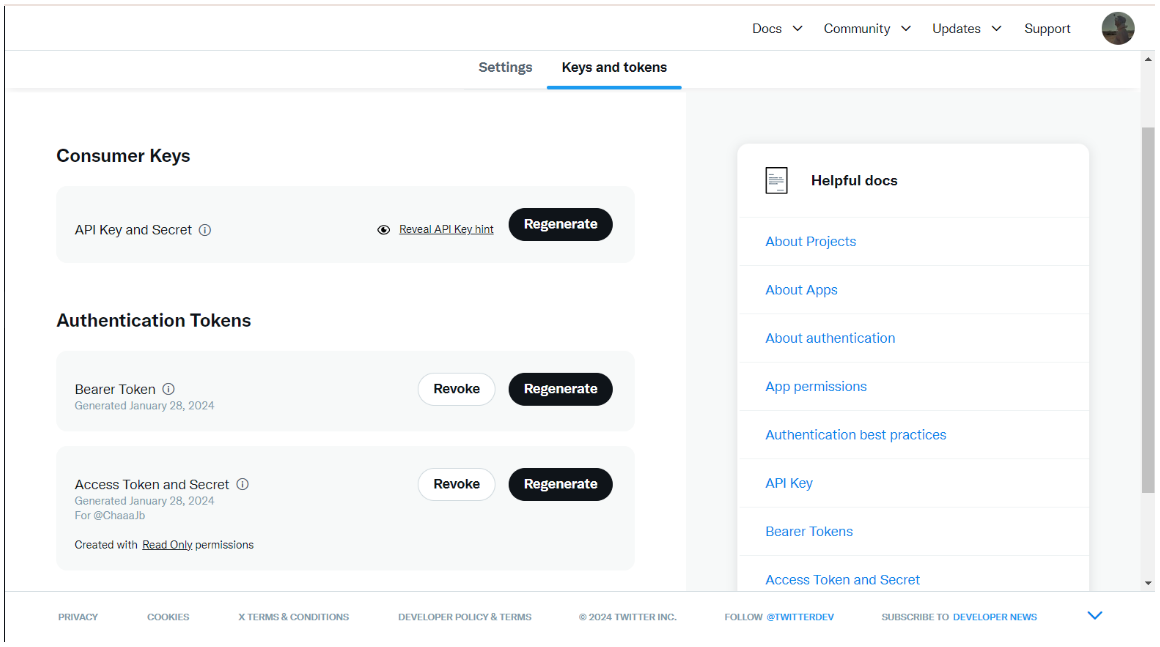
Generic Build failure - check logs for details.

- **Source code implies that the application expects certain environment variables to be set for it to function correctly. These environment variables are related to Twitter's API credentials, which are necessary for authenticating and interacting with the Twitter API.**



```java
public class MyResource {

        User u = new User();
        List<String> tweetsFromUser = new ArrayList<String>();

        try {

            ConfigurationBuilder cb = new ConfigurationBuilder();
            cb.setDebugEnabled(true)
                    .setOAuthConsumerKey(System.getenv("TWITTER_CONSUMER_KEY"))
                    .setOAuthConsumerSecret(
                            System.getenv("TWITTER_CONSUMER_SERVICE"))
                    .setOAuthAccessToken(
                            System.getenv("TWITTER_OAUTH_ACCESS_TOKEN"))
                    .setOAuthAccessTokenSecret(
                            System.getenv("TWITTER_OAUTH_ACCESS_TOKEN_SECRET"));
            TwitterFactory tf = new TwitterFactory(cb.build());
            Twitter twitter = tf.getInstance();
```

➢ **I tried to register the application with Twitter and create an application on the Twitter Developer Platform.**



Consumer Keys

API Key and Secret — Reveal API Key hint — Regenerate

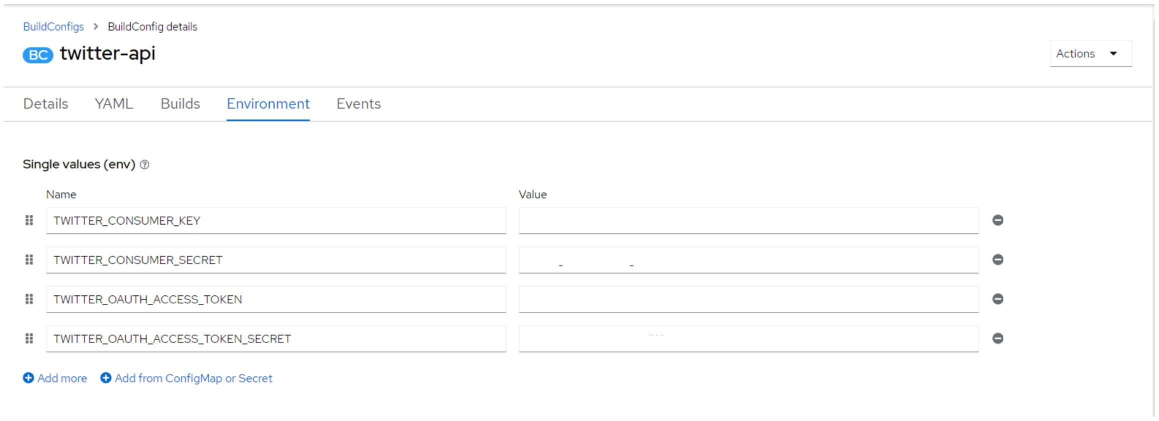
Authentication Tokens

Bearer Token — Generated January 28, 2024 — Revoke — Regenerate

Access Token and Secret — Generated January 28, 2024 — For @ChaaaJb — Created with Read Only permissions — Revoke — Regenerate

Helpful docs
- About Projects
- About Apps
- About authentication
- App permissions
- Authentication best practices
- API Key
- Bearer Tokens
- Access Token and Secret

➢ **FAILED**



Single values (env)

Name	Value
TWITTER_CONSUMER_KEY	
TWITTER_CONSUMER_SECRET	
TWITTER_OAUTH_ACCESS_TOKEN	
TWITTER_OAUTH_ACCESS_TOKEN_SECRET	

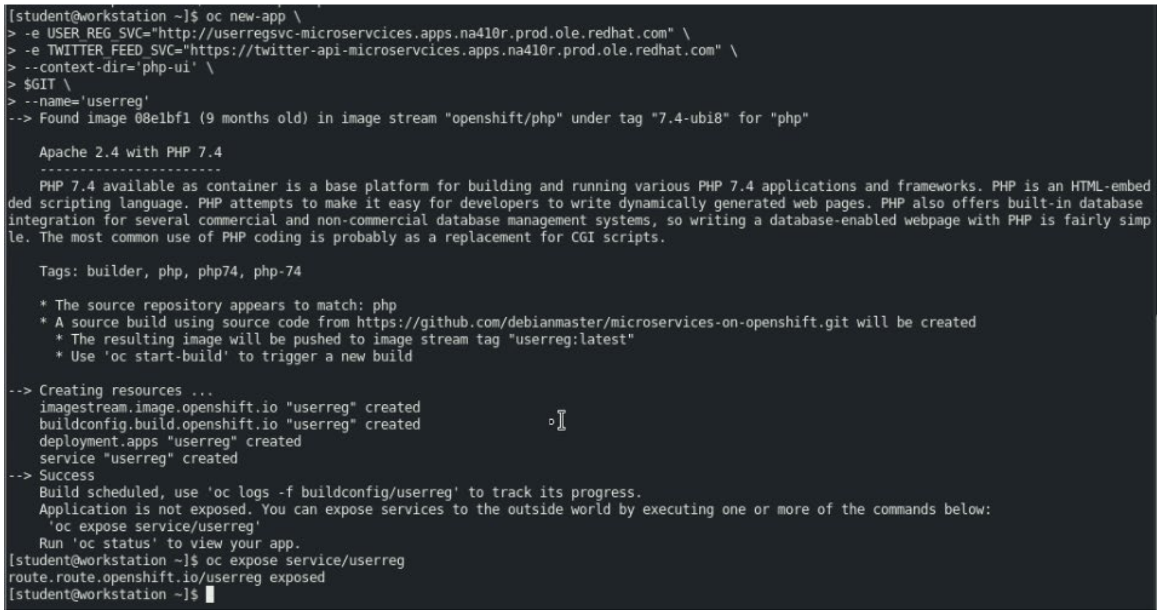
V- Microservice 4: Frontend User Registration Application

Architecture Overview

- **Type**: PHP microservice functioning as the frontend for user registration.
- **Functionality**: Generates HTML and JavaScript code for web browser execution.
- **Communication**: Executes AJAX calls to the backend User Registration service using REST APIs.
- **Environment Variables**: Configures environment variables for backend service access (USER_REG_SVC and TWITTER_FEED_SVC).

Deploy the Frontend User Registration Application

1. **Microservice Deployment**: Deploy from the GitHub repository using **oc new-app**.
2. **Directory Specification**: Indicate the 'php-ui' directory path with the **--context-dir** flag.
3. **Environment Variables**: Set variables for accessing the backend User Registration and Twitter Feeds API services.
4. **Naming and Labeling**: Assign 'userreg' as the name and label it as **microservice=userreg**.



```
[student@workstation ~]$ oc new-app \
> -e USER_REG_SVC="http://userregsvc-microservcices.apps.na410r.prod.ole.redhat.com" \
> -e TWITTER_FEED_SVC="https://twitter-api-microservcices.apps.na410r.prod.ole.redhat.com" \
> --context-dir='php-ui' \
> $GIT \
> --name='userreg'
--> Found image 08e1bf1 (9 months old) in image stream "openshift/php" under tag "7.4-ubi8" for "php"

    Apache 2.4 with PHP 7.4
    -----------------------
    PHP 7.4 available as container is a base platform for building and running various PHP 7.4 applications and frameworks. PHP is an HTML-embed
ded scripting language. PHP attempts to make it easy for developers to write dynamically generated web pages. PHP also offers built-in database
integration for several commercial and non-commercial database management systems, so writing a database-enabled webpage with PHP is fairly simp
le. The most common use of PHP coding is probably as a replacement for CGI scripts.

    Tags: builder, php, php74, php-74

    * The source repository appears to match: php
    * A source build using source code from https://github.com/debianmaster/microservices-on-openshift.git will be created
      * The resulting image will be pushed to image stream tag "userreg:latest"
      * Use 'oc start-build' to trigger a new build

--> Creating resources ...
    imagestream.image.openshift.io "userreg" created
    buildconfig.build.openshift.io "userreg" created
    deployment.apps "userreg" created
    service "userreg" created
--> Success
    Build scheduled, use 'oc logs -f buildconfig/userreg' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose service/userreg'
    Run 'oc status' to view your app.
[student@workstation ~]$ oc expose service/userreg
route.route.openshift.io/userreg exposed
[student@workstation ~]$ 
```

Expose the Frontend Service

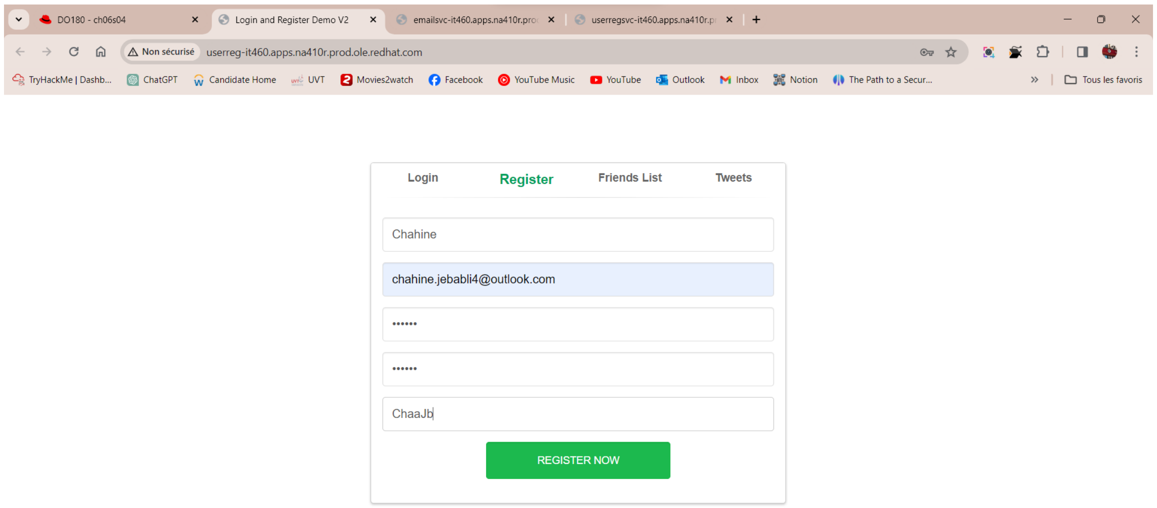
- **URL Generation**: Expose the frontend application as a service using **oc expose** to create a URL for access.

Additional Considerations

- **Inter-Service Communication**: Ensure efficient communication between the frontend application and backend services.
- **Functionality Testing**: Test features like user registration and Twitter feed retrieval.
- **Logging and Monitoring**: Implement strategies for tracking performance and usage.

Communication with Other Services

- The frontend application interacts with the backend User Registration service and the Twitter Feeds API through AJAX calls.



userreg-it460.apps.na410r.prod.ole.redhat.com

Chahine

chahine.jebabli4@outlook.com

••••••

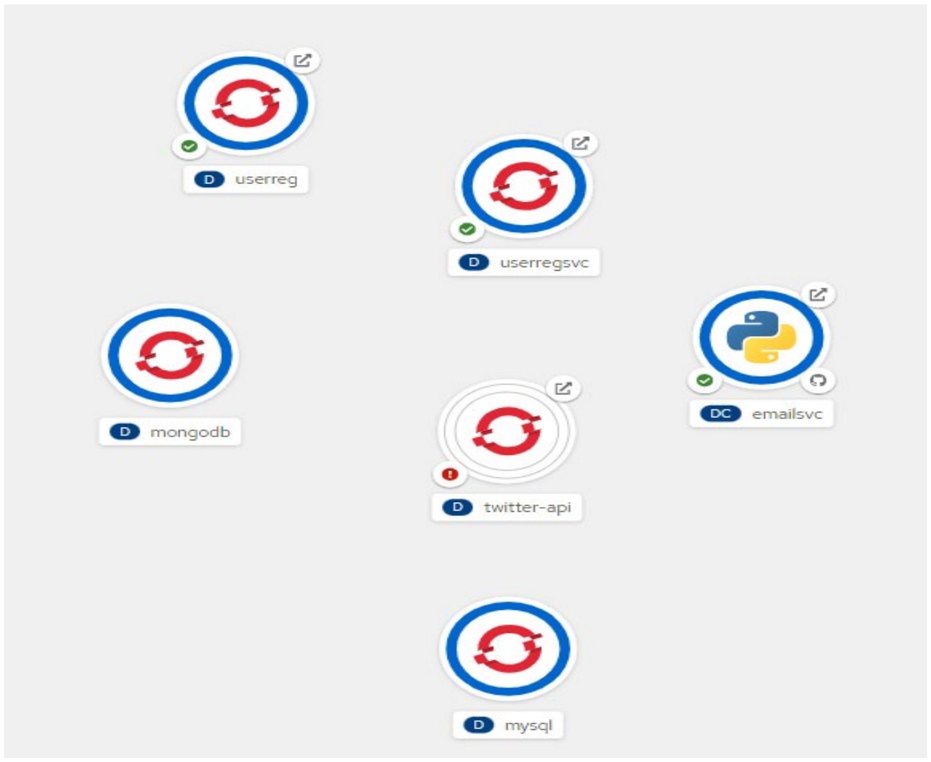
••••••

ChaaJb

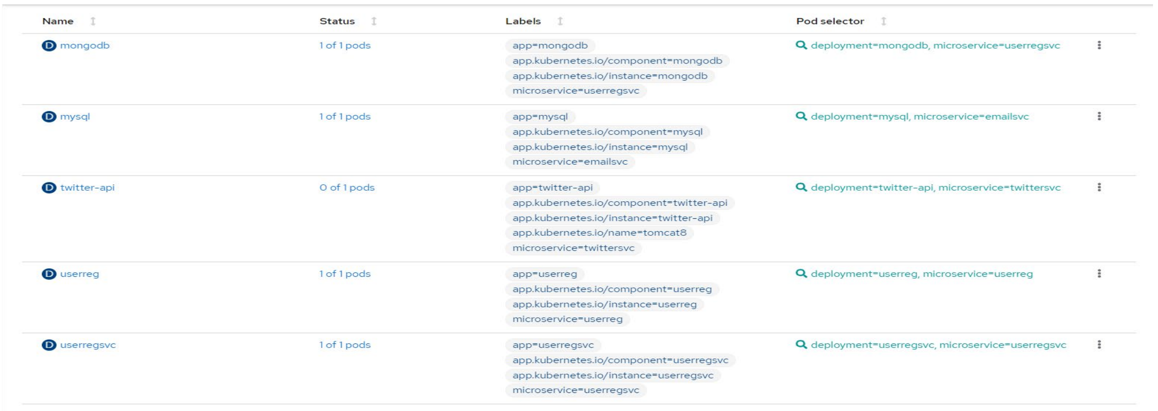
REGISTER NOW

VI- Project Inventory

Topology:



Deployments:



Name	Status	Labels	Pod selector

D mongodb	1 of 1 pods	app=mongodb
		app.kubernetes.io/component=mongodb
		app.kubernetes.io/instance=mongodb
		microservice=userregsvc
			deployment=mongodb, microservice=userregsvc

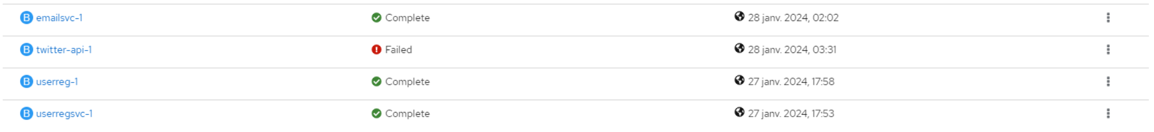
D mysql	1 of 1 pods	app=mysql
		app.kubernetes.io/component=mysql
		app.kubernetes.io/instance=mysql
		microservice=emailsvc
			deployment=mysql, microservice=emailsvc

D twitter-api	0 of 1 pods	app=twitter-api app.kubernetes.io/component=twitter-api app.kubernetes.io/instance=twitter-api app.kubernetes.io/name=tomcat8 microservice=twittersvc	deployment=twitter-api, microservice=twittersvc

D userreg	1 of 1 pods	app=userreg
		app.kubernetes.io/component=userreg
		app.kubernetes.io/instance=userreg
		microservice=userreg
			deployment=userreg, microservice=userreg

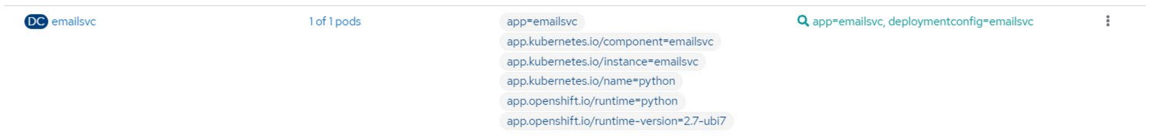
D userregsvc	1 of 1 pods	app=userregsvc
		app.kubernetes.io/component=userregsvc
		app.kubernetes.io/instance=userregsvc
		microservice=userregsvc
			deployment=userregsvc, microservice=userregsvc

Builds:



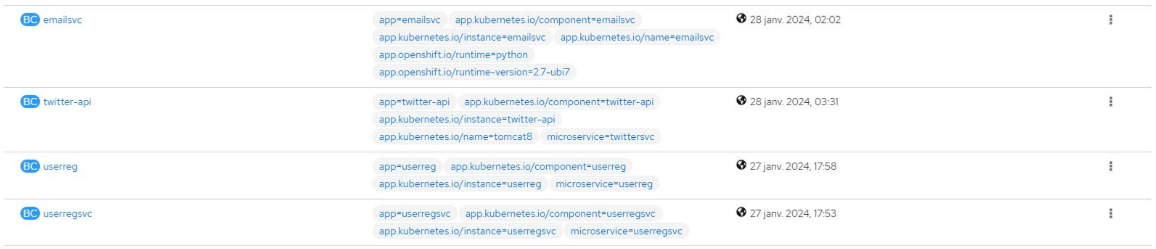
B emailsvc-1	✓ Complete	28 janv. 2024, 02:02
B twitter-api-1	! Failed	28 janv. 2024, 03:31
B userreg-1	✓ Complete	27 janv. 2024, 17:58
B userregsvc-1	✓ Complete	27 janv. 2024, 17:53

DeploymentConfig:



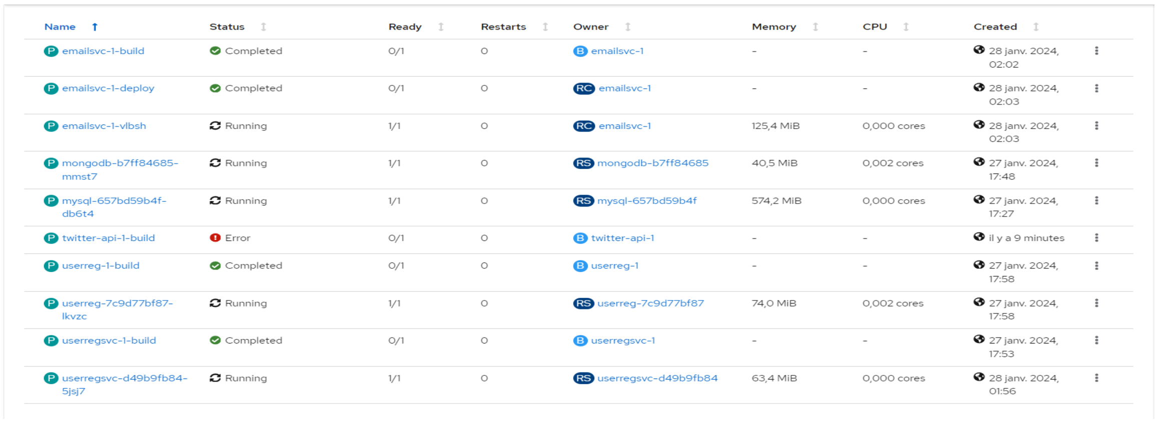
DC emailsvc	1 of 1 pods	app=emailsvc app.kubernetes.io/component=emailsvc app.kubernetes.io/instance=emailsvc app.kubernetes.io/name=python app.openshift.io/runtime=python app.openshift.io/runtime-version=2.7-ubi7	app=emailsvc, deploymentconfig=emailsvc

BuildConfig:



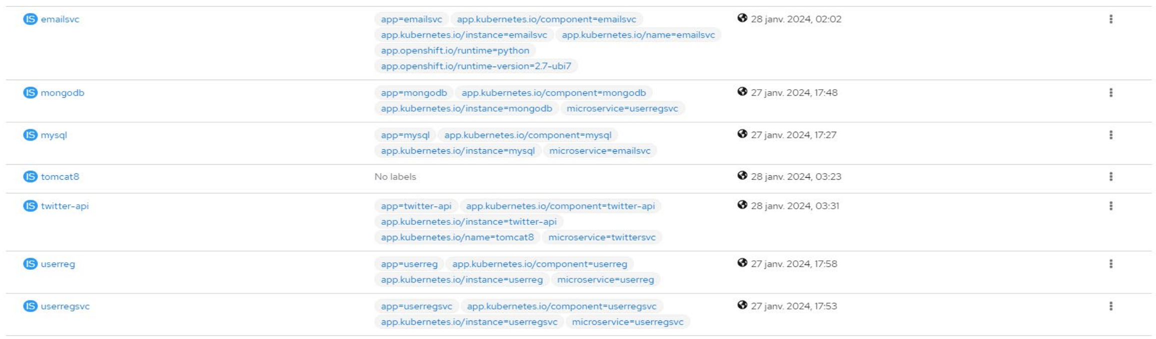
BC emailsvc	app=emailsvc app.kubernetes.io/component=emailsvc app.kubernetes.io/instance=emailsvc app.kubernetes.io/name=emailsvc app.openshift.io/runtime=python app.openshift.io/runtime-version=2.7-ubi7	28 janv. 2024, 02:02
BC twitter-api	app=twitter-api app.kubernetes.io/component=twitter-api app.kubernetes.io/instance=twitter-api app.kubernetes.io/name=tomcat8 microservice=twittersvc	28 janv. 2024, 03:31
BC userreg	app=userreg app.kubernetes.io/component=userreg app.kubernetes.io/instance=userreg microservice=userreg	27 janv. 2024, 17:58
BC userregsvc	app=userregsvc app.kubernetes.io/component=userregsvc app.kubernetes.io/instance=userregsvc microservice=userregsvc	27 janv. 2024, 17:53

Pods:



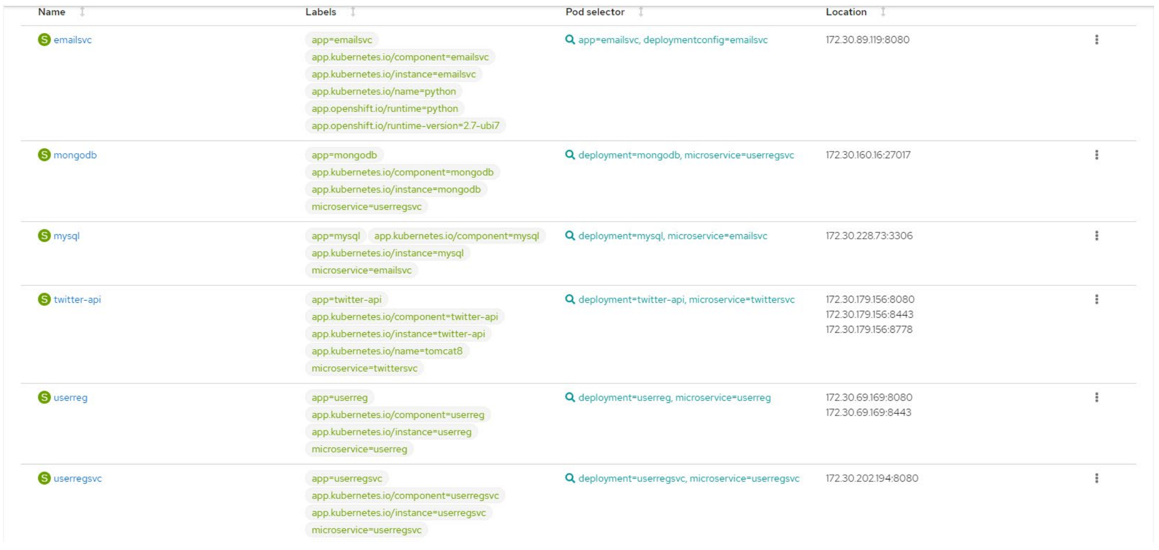
Name	Status	Ready	Restarts	Owner	Memory	CPU	Created
P emailsvc-1-build	Completed	0/1	0	B emailsvc-1	-	-	28 janv. 2024, 02:02
P emailsvc-1-deploy	Completed	0/1	0	RC emailsvc-1	-	-	28 janv. 2024, 02:03
P emailsvc-1-vlbsh	Running	1/1	0	RC emailsvc-1	125,4 MiB	0,000 cores	28 janv. 2024, 02:03
P mongodb-b7ff84685-mmst7	Running	1/1	0	RS mongodb-b7ff84685	40,5 MiB	0,002 cores	27 janv. 2024, 17:48
P mysql-657bd59b4f-db6t4	Running	1/1	0	RS mysql-657bd59b4f	574,2 MiB	0,000 cores	27 janv. 2024, 17:27
P twitter-api-1-build	Error	0/1	0	B twitter-api-1	-	-	il y a 9 minutes
P userreg-1-build	Completed	0/1	0	B userreg-1	-	-	27 janv. 2024, 17:58
P userreg-7c9d77bf87-lkvzc	Running	1/1	0	RS userreg-7c9d77bf87	74,0 MiB	0,002 cores	27 janv. 2024, 17:58
P userregsvc-1-build	Completed	0/1	0	B userregsvc-1	-	-	27 janv. 2024, 17:53
P userregsvc-d49b9fb84-5jsj7	Running	1/1	0	RS userregsvc-d49b9fb84	63,4 MiB	0,000 cores	28 janv. 2024, 01:56

ImageStream:



IS emailsvc	app=emailsvc app.kubernetes.io/component=emailsvc app.kubernetes.io/instance=emailsvc app.kubernetes.io/name=emailsvc app.openshift.io/runtime=python app.openshift.io/runtime-version=2.7-ubi7	28 janv. 2024, 02:02
IS mongodb	app=mongodb app.kubernetes.io/component=mongodb app.kubernetes.io/instance=mongodb microservice=userregsvc	27 janv. 2024, 17:48
IS mysql	app=mysql app.kubernetes.io/component=mysql app.kubernetes.io/instance=mysql microservice=emailsvc	27 janv. 2024, 17:27
IS tomcat8	No labels	28 janv. 2024, 03:23
IS twitter-api	app=twitter-api app.kubernetes.io/component=twitter-api app.kubernetes.io/instance=twitter-api app.kubernetes.io/name=tomcat8 microservice=twittersvc	28 janv. 2024, 03:31
IS userreg	app=userreg app.kubernetes.io/component=userreg app.kubernetes.io/instance=userreg microservice=userreg	27 janv. 2024, 17:58
IS userregsvc	app=userregsvc app.kubernetes.io/component=userregsvc app.kubernetes.io/instance=userregsvc microservice=userregsvc	27 janv. 2024, 17:53

Services:



Name	Labels	Pod selector	Location
emailsvc	app=emailsvc app.kubernetes.io/component=emailsvc app.kubernetes.io/instance=emailsvc app.kubernetes.io/name=python app.openshift.io/runtime=python app.openshift.io/runtime-version=2.7-ubi7	app=emailsvc, deploymentconfig=emailsvc	172.30.89.119:8080

mongodb	app=mongodb
	app.kubernetes.io/component=mongodb
	app.kubernetes.io/instance=mongodb
	microservice=userregsvc
		deployment=mongodb, microservice=userregsvc	172.30.160.16:27017

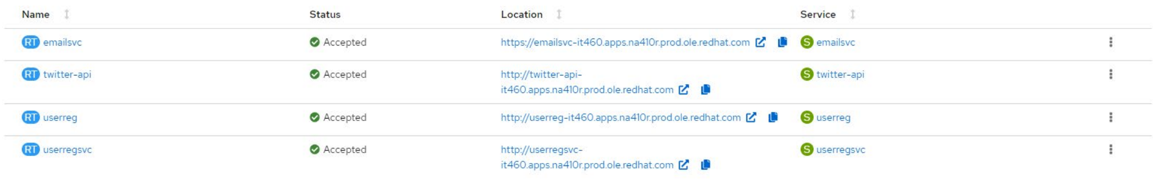
mysql	app=mysql
	app.kubernetes.io/component=mysql
	app.kubernetes.io/instance=mysql
	microservice=emailsvc
		deployment=mysql, microservice=emailsvc	172.30.228.73:3306

twitter-api	app=twitter-api app.kubernetes.io/component=twitter-api app.kubernetes.io/instance=twitter-api app.kubernetes.io/name=tomcat8 microservice=twittersvc	deployment=twitter-api, microservice=twittersvc	172.30.179.156:8080 172.30.179.156:8443 172.30.179.156:8778

userreg	app=userreg
	app.kubernetes.io/component=userreg
	app.kubernetes.io/instance=userreg
	microservice=userreg
		deployment=userreg, microservice=userreg	172.30.69.169:8080
			172.30.69.169:8443

userregsvc	app=userregsvc
	app.kubernetes.io/component=userregsvc
	app.kubernetes.io/instance=userregsvc
	microservice=userregsvc
		deployment=userregsvc, microservice=userregsvc	172.30.202.194:8080

Routes:



Name	Status	Location	Service
emailsvc	Accepted	https://emailsvc-it460.apps.na410r.prod.ole.redhat.com	emailsvc
twitter-api	Accepted	http://twitter-api-it460.apps.na410r.prod.ole.redhat.com	twitter-api
userreg	Accepted	http://userreg-it460.apps.na410r.prod.ole.redhat.com	userreg
userregsvc	Accepted	http://userregsvc-it460.apps.na410r.prod.ole.redhat.com	userregsvc

Conclusion

This report delineates the deployment methodologies, architectural choices, and achieved objectives for each microservice in our project. The microservices are meticulously engineered and deployed to fulfill their designated functions. Efforts have been made to ensure effective communication, scalability, and monitoring within the OpenShift framework. Future enhancements could include improved logging, monitoring, and integration capabilities to further refine our microservices architecture.