

Working on the Analysis and Deployment pipeline of Connect iPaaS and Hire2Retire

Prepared By:
Aishwaryaditya Jha(2018BTECHCSE123)

Faculty Guide:
Dr. Gireesh Kaushik



DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING

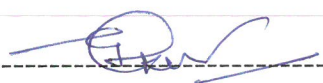
INSTITUTE OF ENGINEERING AND TECHNOLOGY(IET)

JK LAKSHMIPAT UNIVERSITY, JAIPUR

July 2022

CERTIFICATE

This is to certify that the Practice School-2 project work entitled **RoboMQ Pvt. Ltd, Jaipur** submitted by **Aishwaryaditya Jha (2018BTechCSE123)**, towards the partial fulfillment of the requirements for the degree of **Bachelor of Technology in Computer Science and Engineering** of JK Lakshmipat University Jaipur is the record of work carried out by them under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for the Practice School-II examination.



Prof. Gireesh Kaushik

~~Associate~~ Professor,

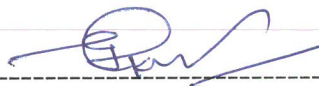
Department of Computer Science

Engineering

Institute of Engineering & Technology

(IET)

JK Lakshmipat University, Jaipur



Dr. Gireesh Kaushik

Ps-2 Co-ordinator,

Department of Computer Science

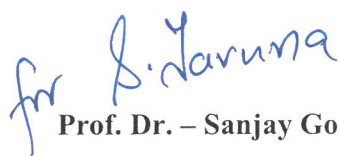
Engineering

Institute of Engineering & Technology

(IET)

JK Lakshmipat University, Jaipur

Date of Submission: 08-07-2022



Prof. Dr. – Sanjay Goel

Director

Institute of Engineering & Technology JK

Lakshmipat University Jaipur

ABOUT THE ORGANISATION

- RoboMQ is an API and data integration company behind some of the leading-edge technologies and platforms that enable enterprises to build business processes and automate their operations using best of breed cloud, SaaS, and on-premise applications that they already own.
- It is a Message Queue as Service platform hosted on cloud and available as an Enterprise hosting option. This Software as a Service platform is an integrated message queue hub, analytics engine, management console, dashboard and monitoring & alerts; all managed and hosted in a secure, reliable and redundant infrastructure.
- RoboMQ platform is an expandable platform that allows addition of business specific application components which feed on the same stream of information, in parallel.
- RoboMQ comes bundled with few application components like monitoring dashboard for messaging activity, data driven analytics and alerts.
- RoboMQ accelerates digital transformation to create competitive advantages using best of the breed SaaS, cloud, on-premise apps, and IoT. Our Connect iPaaS and Hybrid Integration Platform (HIP) provide an intuitive, UX based API and data integration solution for line of business and technology users.
- At RoboMQ, we are constantly innovating to create products that generate greater value for our customers through better design, and reduced time to market. We also provide the advantage of technology advances in creating products with a simple, easy and intuitive user experience (UX).
- Similar applications can be added to expand the platform. For each application component, multiple processes or workers can be added to handle increased demand or load.
- Our Hybrid Integration Platform (HIP) provides advanced integration capabilities using Kubernetes based service mesh, service bus, and API management capabilities for highly distributed and scalable integration of Cloud, SaaS and on-premise applications in a hybrid and federated cloud setup. Integration workflows can be built leveraging prebuilt smart containers and capabilities & frameworks provided by the platform.
- Hybrid Integration Platform (HIP) is a low code offering targeted at integration experts or IT and technology users.
- IoT is revolutionizing enterprises and businesses with availability of information through devices and sensors that need to be processed at the edge and the fog and be made available to business workflows.

Table of Contents

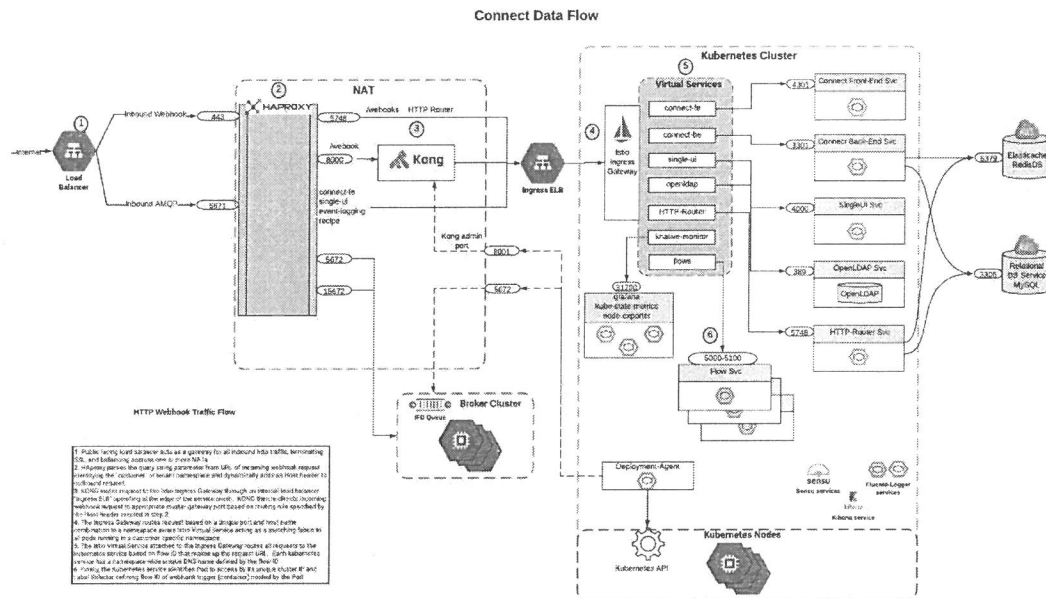
1. Company's Overview	6
2. Company's Vision	6
3. History	6
4. Job Description	7
5. Software Design Specification	7
6. Software Requirements Specification	8
6.1. Purpose	8
6.2. Product Scope	8
6.2.1. Connect iPaaS	8
6.2.2. PIP	9
6.2.3. Hire2Retire	10
6.3. Overall Description	10
6.4. Product Perspective	10
6.5. Product Functions	10
6.6. User Classes and Characteristics	12
6.7. Hardware Interfaces	13
6.8. Software Interfaces	14
6.9. Communications Interfaces	15
6.12. Nonfunctional Requirements	16
6.13. Performance Requirements	16
6.14. Safety and Security requirement	16
6.15. Objectives	16
7. CI/CD Pipeline	17
8. Implementation (algorithm/flow chart) and output screens:	20
8.1. Deploying a new environment for testing:	20
8.2. Applying Templating through Helm Charts:	21
8.3. Using Kubernetes Secrets for applying encoding to the sensitive credentials included in the ConfigMaps	21
8.4. Using Jenkins to configure/schedule build jobs and other processes	38
8.5. Job configurations for company server	47
9. Future Enhancements	51
10. References	51

4. Job Description

A My role as DevOps Engineer includes working on the backend and the infrastructural operation side of the product. I had to work on various DevOps Tools and Frameworks such as Docker, Kubernetes, Jenkins, Terraform. While working with new technologies I had to make sure of feasibility with product and proper and complete use of functions provided by framework or technology for better enhancement of product. The company had been working with the waterfall model in the past, and the newly created DevOps team is assigned the task to automate all the redundant infrastructure tasks, to reduce average deployment and incident recovery time. As DevOps Engineers, we must follow best practices and follow proper conventions while developing and deploying applications. In addition, to provide the best product to clients using latest technologies and functionalities that market has to offer

5. Software Design Specification

System Architecture



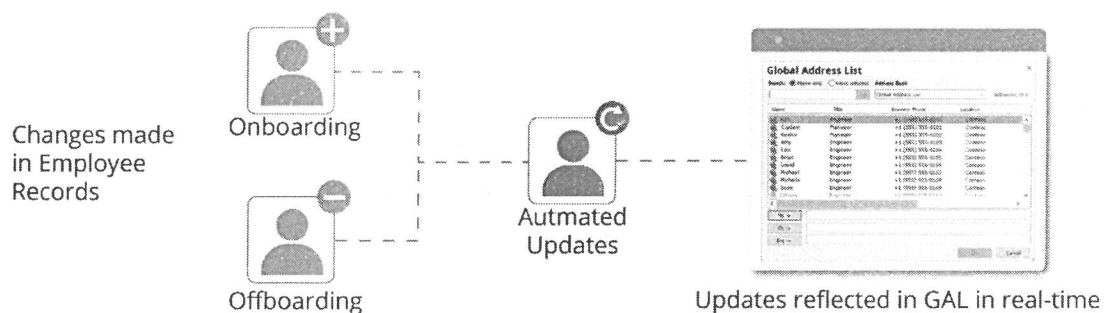
interface which is very easy to use and very easy to make connections within each other. Connect iPaaS is one of the world's leading platforms for providing simple solutions for api and data integration, even for those who are not from technical backgrounds and that is what makes connect very useful and beneficial for any business or individuals. Connect was RoboMQ's first interactive platform where users can integrate two ecosystems on their own by just using simple UI, which made it much simpler for users to Integrate two eco system with no code scheme. Although Connect is RoboMQ's well established platform and already so many users using it, still RoboMQ is doing research and adjustments according to their different client new demands on Connect.

6.2.2. PIP

PIPs are pre-packaged integration solutions that are built as a service (iSaaS) to solve common business problems. Many companies choose to go with a PIP as these are fully managed, operated, and hosted by RoboMQ, which allows customers to be more hands-off. Once the PIP is running, then everything is on autopilot, which frees up more time for customers to focus on other core areas of their business.



Managing GAL is Easier Than You Think!



www.robomq.io

Hire2Retire

sales@robomq.io

Flow Details –

Flow Overview – Suppose the flow what I created in Connect iPaaS is an ecosystem between Salesforce and ServiceNow. The trigger is happening from Salesforce application for the condition whenever a new account object is created with some filters within it. So, if these conditions matches with any new entry, then it will just trigger the action which has been declared in the action section. Flow Usage can be explained as suppose I am running a business and I am using Salesforce and ServiceNow both. So, if some new account comes in Salesforce, then I want that account details also gets stored in some specified ServiceNow table, so I can also have a track for some relevant data in ServiceNow as well without doing any extra manual work. This gives us an ease and time saving benefits.

Trigger Filters are the conditions which needs to be satisfied on the trigger event for firing the action event. I have used filter like whenever the account type is Customer – Direct and Industry is Engineering or whenever the account name contains the work “Gupta”.

Action Filters are the condition which needs to be satisfied on the action initializing side for the action to fire. Action filters have been created for in case when multiple action needs to be taken (for multiple applications) and when we have to set different conditions/filters to different action applications.

Transformer is a tab on connect where we set the functionality that needs to happen when action event fires. In Transformer there has been given different functions and features to use relevant data and functionality. I am forming a column which will contain account name , account number , account id, account phone and customer priority in ServiceNow specified table.

Deploy is an event which we can fire when our flow has been created successfully for seeing the change in a practical manner.

Workflow is the full flow of working events (trigger and action) and hence performing the connection.

Flow Summary is the overall details in brief of what flow has been created and how it is functioning. Recipes are something we can say like all already created flow which are relevant and can be useful for individuals and different business organizations.

Hire2Retire :-

Activities Firefox Web Browser Mar 16, 12:56

Hire2Retire Pre-Package x

https://beta.robomq.io/hire2retire/flow/a1c68d3b-be08-486e-8897-81b2975274d9/design

TEST Draft

Overview Design Connections Observe

1 HR Employee Data Definition

2 Employment Lifecycle Sync

3 Employment Status

4 HR to AD Profile Map

5 Organizational Unit Assignment

6 Security Group Assignment (Optional)

7 Distribution List Assignment (Optional)

HR Employee Data Definition

CSV filename Pattern to Scan *

animesh .CSV

Provide the source file name (e.g., Employee_Report) or a valid regex (e.g., Employee_*.Report). The process will automatically append the ".CSV" extension.

Provide header names in your CSV file

Header Name	Header Type
NAME	Text
CLASS	Text
AGE	Text
ROLE	Text

Apply Filter

6.7. Hardware Interfaces

The following off-premises hardware providers are used for hosting the above products.

- **Amazon Elastic Compute Cloud (Amazon EC2)** offers the broadest and deepest compute platform, with over 500 instances and choice of the latest processor, storage, networking, operating system, and purchase model to help you best match the needs of your workload. With EC2, our company can provision several virtual computers—depending on their immediate processing needs—to crunch numbers and only have to pay for what they use.
- **AWS ECR:** Amazon Elastic Container Registry (Amazon ECR) is an AWS managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM. This is so that our Amazon EC2 instances can access our container repositories and images. We use Ubuntu CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

- **Git:** Git is a DevOps tool used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development.
- **Terraform:** HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all your infrastructure throughout its lifecycle.
- **Helm:** In a vast infrastructure with dozens or hundreds of microservices, with each microservice having their own YAML files, it can be a cumbersome process to deploy each of them manually so it basically helps in managing them.

6.9. Communications Interfaces

- **AWS ELB:** Elastic Load Balancing automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It monitors the health of its registered targets, and routes traffic only to the healthy targets. Elastic Load Balancing scales your load balancer as your incoming traffic changes over time. It can automatically scale to the vast majority of workloads.

6.10. AWS VPC: Amazon Virtual Private Cloud (Amazon VPC) gives you full control over your virtual networking environment, including resource placement, connectivity, and security. Get started by setting up your VPC in the AWS service console. Next, add resources to it such as Amazon Elastic Compute Cloud (EC2) and Amazon Relational Database Service (RDS) instances. Finally, define how your VPCs communicate with each other across accounts, Availability Zones, or AWS Regions. In the example below, network traffic is being shared between two VPCs within each Region.

6.11. Functional Requirements

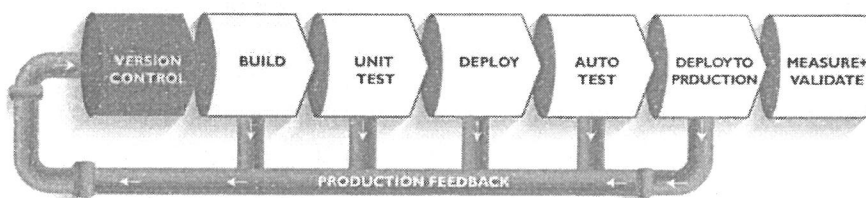
- The code should be built automatically as soon as a developer pushes the code on the company repository
- A notification should be sent to the senior developers of all teams when the system detects any kind of a failure
- The automated pipeline provide standardized feedback loops to developers, and enable fast product iterations.

7. CI/CD Pipeline

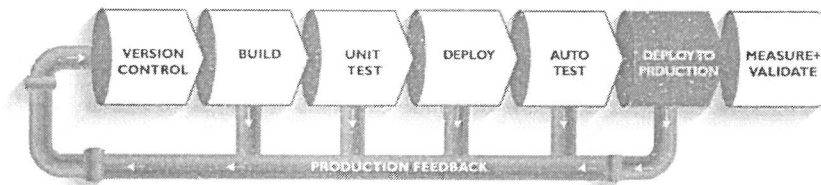
A CI/CD Pipeline implementation, or Continuous Integration/Continuous Deployment, is the backbone of the modern DevOps environment. It bridges the gap between development and operations teams by automating the building, testing, and deployment of applications. In this blog, we will learn what a CI/CD pipeline is and how it works. DevOps is a software development approach which involves continuous development, continuous testing, continuous integration, continuous deployment, and continuous monitoring of the software throughout its development lifecycle. This is the process adopted by all the top companies to develop high-quality software and shorter development lifecycles, resulting in greater customer satisfaction, something that every company wants.

A CI/CD pipeline automates your software delivery process. The pipeline builds code, runs tests (CI), and safely deploys a new version of the application (CD). Automated pipelines remove manual errors, provide standardized feedback loops to developers, and enable fast product iterations.

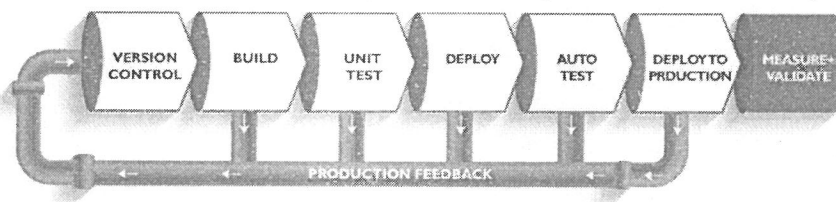
CI, short for Continuous Integration, is a software development practice in which all developers merge code changes in a central repository multiple times a day. CD stands for Continuous Delivery, which on top of Continuous Integration adds the practice of automating the entire software release process. With CI, each change in code triggers an automated build-and-test sequence for the given project, providing feedback to the developer(s) who made the change. The entire CI feedback loop should run in less than 10 minutes. Continuous Delivery includes infrastructure provisioning and deployment, which may be manual and consist of multiple stages. What's important is that all these processes are fully automated, with each run fully logged and visible to the entire team.



The above pipeline is a logical demonstration of how software will move along the various stages in this lifecycle before it is delivered to the customer or before it is live in production. When we build a web application which is going to be deployed on live web servers, we have a set of developers responsible for writing the code, who will further go on and build the web application, then this code is committed into a version control system (such as git, svn) by the team of developers. Next, it goes through the build phase, which is the first phase of the pipeline, where developers put in their code and then again the code goes to the version control system with a proper version tag.



Meanwhile, in every step, if there is an error, we can shoot an email back to the development team so that they can fix it. Then they will push it into the version control system and it goes back into the pipeline. Once again, if there is any error reported during testing, the feedback goes to the dev team again, where they fix it and the process reiterates if required



This lifecycle continues until we get code/a product which can be deployed to the production server where we measure and validate the code. We now understand the CI/CD Pipeline and its working; now, we will move on to understand what Jenkins is and how we can deploy the demonstrated code using Jenkins and automate the entire process

Additional benefits of pipelines

Having a CI/CD pipeline has more positive effects than only making an existing process a little more efficient:

- ☐ Developers can stay focused on writing code and monitoring the behavior of the system in production.
- ☐ QA and product stakeholders have easy access to the latest, or any, version of the system.
- ☐ Product updates are not stressful.
- ☐ Logs of all code changes, tests and deployments are available for inspection at any time.
- ☐ Rolling back to a previous version in the event of a problem is a routine push-button action.
- ☐ A fast feedback loop helps build an organizational culture of learning and responsibility.

8.2. Applying Templating through Helm Charts:

- Listing all the environmental variables (including versioning, credentials, etc.) required categorized by Application/Plugin in Helm Values.yaml
- Creating multiple templates grouped by similar configuration files, and mapping the environment variables in it
- Using public charts pulled from ArtifactHub and BitNami to fast track the process
- Configuring sub charts for front-end, back-end, EventLogging, etc.

```
ayushrathore@DESKTOP-AR607FQ:~$ helm install demo5 ConnectSt/
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/ayushrathore/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/ayushrathore/.kube/config
NAME: demo5
LAST DEPLOYED: Thu May 5 14:43:15 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=ConnectSt,app.kubernetes.io/instance=demo5" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8686 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8686:$CONTAINER_PORT
```

8.3. Using Kubernetes Secrets for applying encoding to the sensitive credentials included in the ConfigMaps

- To create the Secret containing the **REDIS_ROOT_PASSWORD**, choose a password and convert it to base64:

```
# The root password will be "U8y743i"
$ echo -n 'U8y743i' | base64
S3ViZXJuZXRlc1JvY2tzIQ==
```

- Make a note of the encoded string. We need it to create the YAML file for the Secret:

password: 16 bytes

Note that the **Data** field contains the key you set in the YAML: **password**. The value assigned to that key is the password you created, but it is not shown in the output. Instead, the value's size is shown in its place, in this case, 16 bytes.

You can also use the **kubectl edit secret <secretname>** command to view and edit the Secret. If you edit the Secret, you'll see something like this:

```
apiVersion: v1
data:
  password: S3ViZXJuZXRlc1JvY2tzIQ==
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
{"apiVersion":"v1","data":{"password":"S3ViZXJuZXRlc1JvY2tzIQ=="},
"kind":"Secret","metadata":{"annotations":{},"name":"robomq-redis-password",
"namespace":"secrets-and-configmaps"},"type":"Opaque"}
  creationTimestamp: 2019-05-29T12:06:09Z
  name: robomq-redis-password
  namespace: secrets-and-configmaps
  resourceVersion: "85154772"
  selfLink: /api/v1/namespaces/secrets-and-configmaps/secrets/robomq-redis-password
  uid: 2542dadb-820a-11e9-ac24-005056a1db05
type: Opaque
```

ConfigMaps

Secrets and ConfigMaps are comparable. In the same way, they may be produced and shared in containers. The base64-encoding obfuscation is the only significant variation between them.

ConfigMaps are an excellent way to generate customised running services from generic container images and are intended for non-sensitive data—configuration data—like config files and environment variables.

Creating a ConfigMap

Secrets and ConfigMaps can both be produced in the same way. We can either manually write a YAML representation of the ConfigMap and load it into Kubernetes, or use the `kubectl create configmap` command from the command line. Instead of giving literal strings (like with `--from-literal=key=>string>` in the Secret above), the next task generates a ConfigMap from an existing file—a Redis config intended for `/etc/redis/conf.d` in the container—using the later way. This config file overrides Robomqdb's default setting of 16M for max allowed packet.

First, create a file named **max_allowed_packet.cnf** with the following content:

```
[redisd]
max_allowed_packet = 64M
```

This will override the default setting in the **my.cnf** file and set **max_allowed_packet** to 64M.

Once the file is created, We can create a ConfigMap named **robomqdb-config** using the **kubectl create configmap** command that contains the file:

```
$ kubectl create configmap robomqdb-config --from-file=max_allowed_packet.cnf
configmap/robomqdb-config created
```

Just like Secrets, ConfigMaps store one or more key/value pairs in their Data hash of the object. By default, using **--from-file=<filename>** (as above) will store the contents of the file as the value, and the name of the file will be stored as the key. This is convenient from an organization viewpoint. However, the key name can be explicitly set, too. For example, if we used **--from-file=max-packet=max_allowed_packet.cnf** when we created the ConfigMap, the key would be **max-packet** rather than the file name. If we had multiple files to store in the ConfigMap, we could add each of them with an additional **--from-file=<filename>** argument.

View the new ConfigMap and read the data

As mentioned, ConfigMaps are not meant to store sensitive data, so the data is not encoded when the ConfigMap is created. This makes it easy to view and validate the data and edit it directly.

First, validate that the ConfigMap was, indeed, created:

```
$ kubectl get configmap robomqdb-config
NAME          DATA      AGE
robomqdb-config 1         9m
```

The `kubectl describe` command can be used to inspect the contents of the ConfigMap. It's worth noting that the entire contents of the file are visible, and the key name is, in fact, `max_allowed_packet.cnf`.

```
selfLink: /api/v1/namespaces/secrets-and-configmaps/configmaps/robomqdb-config
uid: c83ccfae-82d2-11e9-832f-005056a1102f
```

After saving the change, verify the data has been updated:

```
# Note the '.' in max_allowed_packet.cnf needs to be escaped
$ kubectl get configmap robomqdb-config -o "jsonpath={.data['max_allowed_packet\.cnf']}"
```

```
[redisd]
max_allowed_packet = 32M
```

Using Secrets and ConfigMaps

Secrets and ConfigMaps can be mounted in a container as environment variables or files. The Secrets as environment variables and the ConfigMap as a file must be mounted for the Robomqdb container. However, we must first create a Robomqdb Deployment in order to interact with it. Create a file named **robomqdb-deployment.yaml** with the following:

```
apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    app: robomqdb

  name: robomqdb-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: robomqdb

  template:

    metadata:

      labels:
```

secretKeyRef:

name: robomq-redis-password

key: password

Note that the name of the object is the name of the environment variable that is added to the container. The **valueFrom** field defines **secretKeyRef** as the source from which the environment variable will be set; i.e., it will use the value from the **password** key in the **robomq-redis-password** Secret we set earlier.

Add this section to the definition for the **robomqdb** container in the **robomqdb-deployment.yaml** file. It should look something like this:

spec:

containers:

- name: robomqdb

image: 70932010.dkr.ecr.us-east-1.amazonaws.com/robomqdb:10.4

env:

- name: REDIS_ROOT_PASSWORD

valueFrom:

secretKeyRef:

name: robomq-redis-password

key: password

ports:

- containerPort: 3306

protocol: TCP

volumeMounts:

- mountPath: /var/lib/redis

name: robomqdb-volume-1

name: robomqdb-user-creds

ports:

- containerPort: 3306

protocol: TCP

volumeMounts:

- mountPath: /var/lib/redis

name: robomqdb-volume-1

Note: We could have just added the redis-root-password Secret to the envFrom list and allowed it to be parsed, provided the password key was named REDIS_ROOT_PASSWORD instead. Unlike env, there is no ability to directly define the name of an environment variable with envFrom.

Add the max_allowed_packet.cnf file to the Deployment as a volumeMount

Both env and envFrom can be used to share ConfigMap key/value pairs with containers, as previously indicated. However, in the case of the **robomqdb-config** ConfigMap, the entire file is stored as the value to the key, and the file needs to exist in the container's filesystem for Robomqdb to be able to use it. Luckily, both Secrets and ConfigMaps can be the source of Kubernetes "volumes" and mounted into the containers instead of using a filesystem or block device as the volume to be mounted.

The **robomqdb-deployment.yaml** already has a volume and volumeMount specified, an **emptyDir** (effectively a temporary or ephemeral) volume mounted to **/var/lib/redis** to store the Robomqdb data:

<...>

volumeMounts:

- mountPath: /var/lib/redis

name: robomqdb-volume-1

<...>

name: robomqdb-config

items:

- key: max_allowed_packet.cnf

path: max_allowed_packet.cnf

name: robomqdb-config-volume

<...>

- The **volumeMount** is pretty self-explanatory—create a volume mount for the **robomqdb-config-volume** (specified in the **volumes** list below it) to the path **/etc/redis/conf.d**.
- Then, in the **volumes** list, **configMap** tells Kubernetes to use the **robomqdb-config** ConfigMap, taking the contents of the key **max_allowed_packet.cnf** and mounting it to the path **max_allowed_packed.cnf**. The name of the volume is **robomqdb-config-volume**, which was referenced in the **volumeMounts** above.
- Note: The **path** from the **configMap** is the name of a file that will contain the contents of the key's value. In this case, our key was a file name, too, but it doesn't have to be. Note also that **items** is a list, so multiple keys can be referenced and their values mounted as files. These files will all be created in the **mountPath** of the **volumeMount** specified above: **/etc/redis/conf.d**.
- **Create a Robomqdb instance from the Deployment**
- At this point, we should have enough to create a Robomqdb instance. We have two Secrets, one holding the **REDIS_ROOT_PASSWORD** and another storing the **REDIS_USER**, and the **REDIS_PASSWORD** environment variables to be added to the container. We also have a ConfigMap holding the contents of a Redis config file that overrides the **max_allowed_packed** value from its default setting.
- We also have a **robomqdb-deployment.yaml** file that describes a Kubernetes deployment of a Pod with a Robomqdb container and adds the Secrets as environment variables and the ConfigMap as a volume-mounted file in the container. It should look like this:

apiVersion: apps/v1

kind: Deployment

metadata:

```
- secretRef:
  name: robomqdb-user-creds

ports:
- containerPort: 3306

protocol: TCP

volumeMounts:
- mountPath: /var/lib/redis
  name: robomqdb-volume-1
- mountPath: /etc/redis/conf.d
  name: robomqdb-config-volume

volumes:
- emptyDir: {}
  name: robomqdb-volume-1
- configMap:
  name: robomqdb-config
  items:
  - key: max_allowed_packet.cnf
    path: max_allowed_packet.cnf
  name: robomqdb-config-volume
```

Check that the root password was set correctly

\$ redis -uroot -p\${REDIS_ROOT_PASSWORD} -e 'show databases;'

```
+-----+
| Database |
+-----+
| information_schema |
| redis |
| performance_schema |
+-----+
```

Check that the max_allowed_packet.cnf was parsed

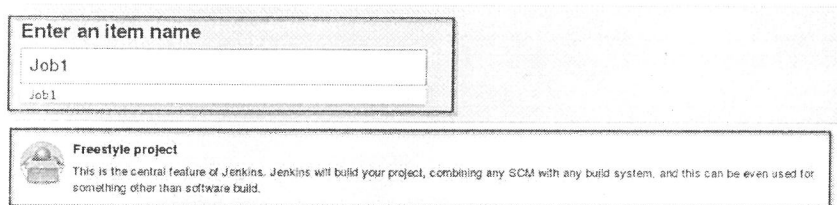
\$ redis -uroot -p\${REDIS_ROOT_PASSWORD} -e "SHOW VARIABLES LIKE 'max_allowed_packet';"

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_allowed_packet | 33554432 |
+-----+-----+
```

Advantages of Secrets and ConfigMaps

After executing this task I learned how to create Kubernetes Secrets and ConfigMaps and how to use those Secrets and ConfigMaps by adding them as environment variables or files inside of a running container instance. This makes it easy to keep the configuration of individual instances of containers separate from the container image. By separating the configuration data, overhead is reduced to maintaining only a single image for a specific type of instance while retaining the flexibility to create instances with a wide variety of configurations.

Step 3: Select a freestyle project and provide the item name (here I have given Job1) and click OK.



Enter an item name

Job1

Job1

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Step 4: Select Source Code Management and provide the Git repository. Click on the Apply and Save button.



General **Source Code Management** Build Triggers Build Environment Build Post-build Actions

Source Code Management

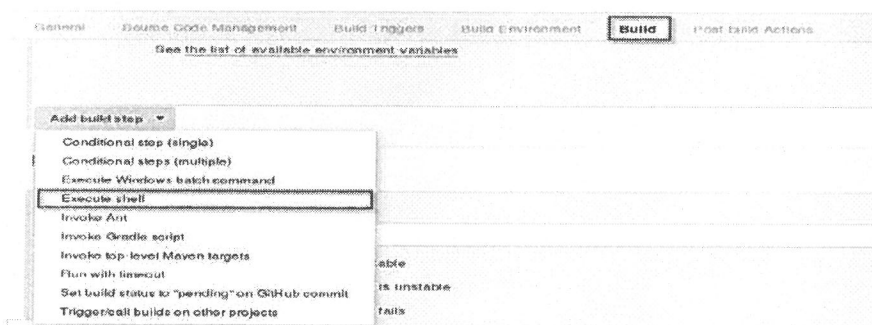
☐ None
☒ Git

Repositories

Repository URL:

Credentials:

Step 5: Then click on Build->Select Execute Shell.



General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

See the list of available environment variables

Add build step

- Conditional step (single)
- Conditional steps (multiple)
- Execute Windows batch command
- Execute shell**
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit
- Trigger/call builds on other projects

Step 6: Provide the shell commands. Here, it will build the archive file to get a war file. After that, it will get the code which is already pulled and then it uses maven to install the package. It simply installs the dependencies and compiles the application.

Enter an item name

Job2

Job2

 **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Step 9: Select Source Code Management and provide the Git repository. Click on the Apply and Save button.

General **Source Code Management** Build Triggers Build Environment Build Post-build Actions

Source Code Management

☐ None
☒ **Git**

Repositories

Repository URL

Credentials [Add](#)

[Advanced...](#)

[Add Repository](#)

Step 10: Then click on Build->Select Execute Shell.

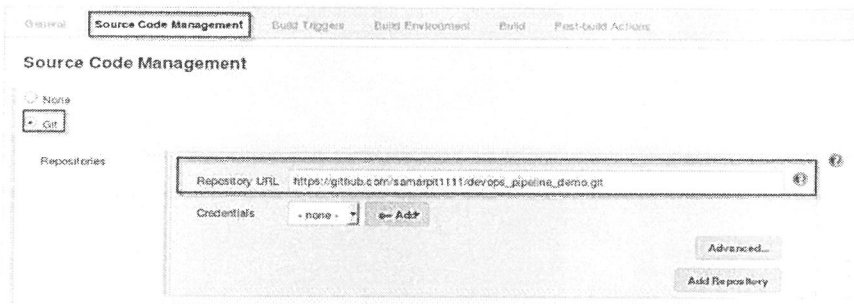
General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

See the list of available environment variables

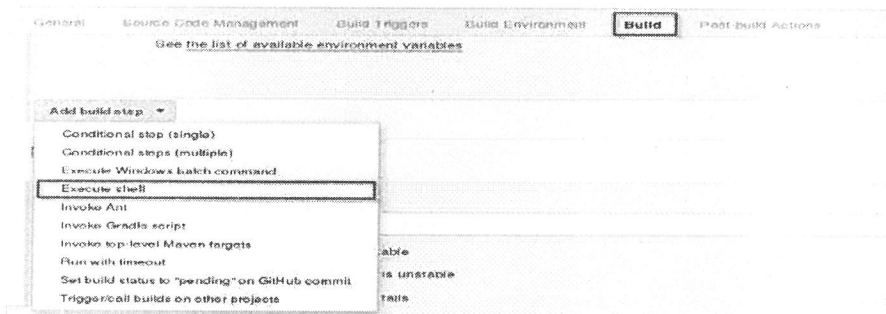
Add build step ▼

- Conditional step (single)
- Conditional steps (multiple)
- Execute Windows batch command
- Execute shell**
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit
- Trigger/call builds on other projects

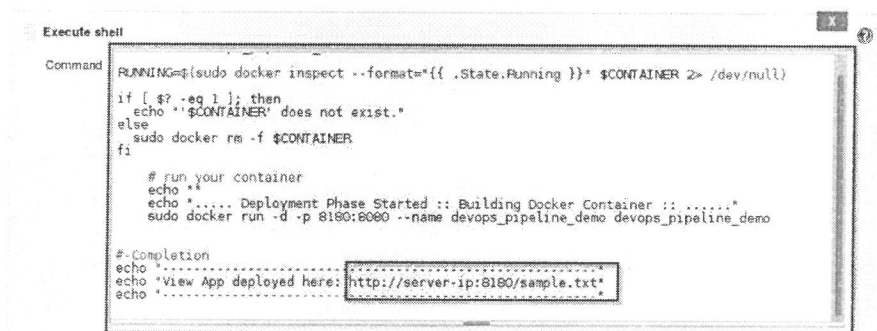
Step 11: Provide the shell commands. Here it will start the integration phase and build the Docker Container.



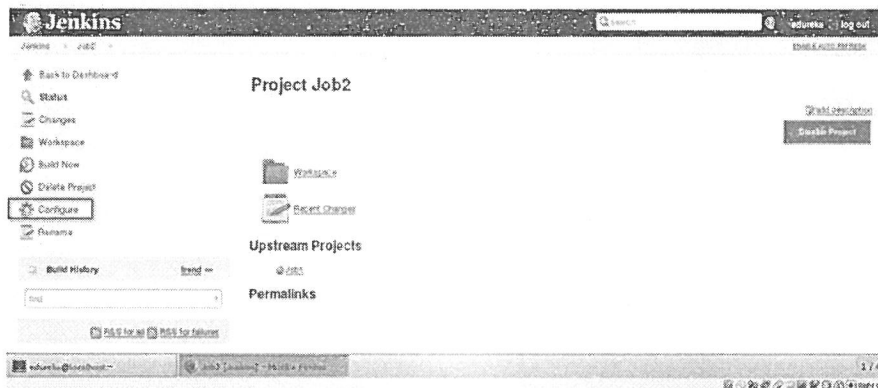
Step 15: Then click on Build->Select Execute Shell.



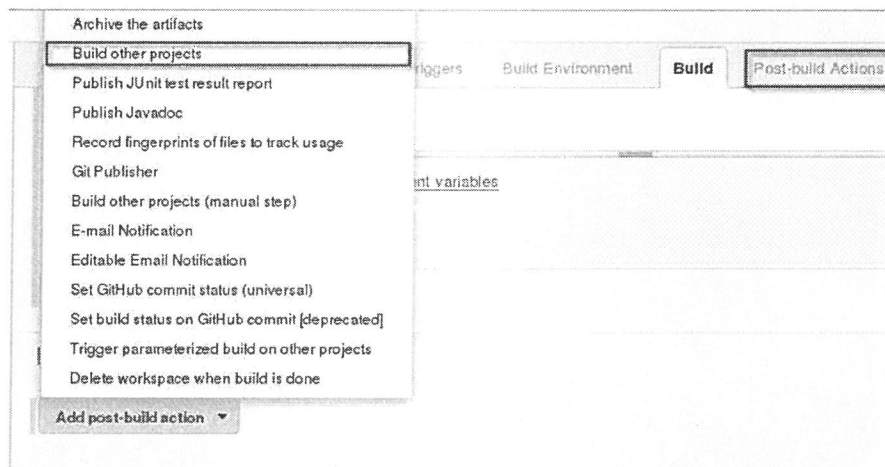
Step 16: Provide the shell commands. Here it will check for the Docker Container file and then deploy it on port number 8180. Click on the Save button.



Step 17: Now click on Job1 -> Configure.

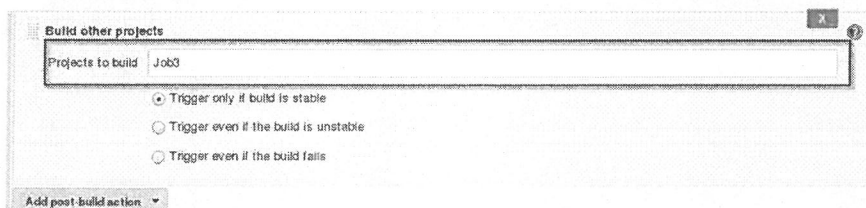


Step 21: Click on Post-build Actions -> Build other projects.

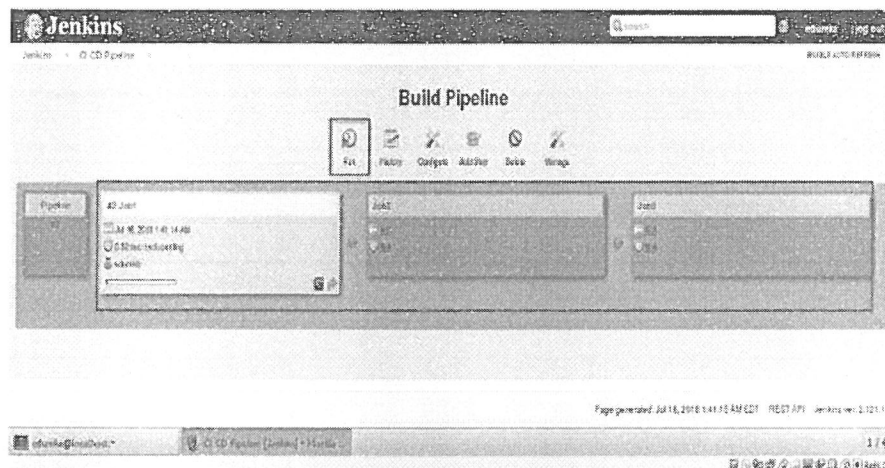


Step 22: Provide the project name to build after Job2 (here is Job3) and then click on Save.

Post-build Actions



Step 26: Click on Run button to start the CI/CD process.



8.5. Job configurations for company server

Postman collection for all the jobs

- Creating a postman collection post-build action will allow users to remotely trigger the build for all the jobs using a REST API (Applications and programming interfaces) call.
- User can import this collection using postman and run the jobs by providing the required parameters
- The environment variables that user needs to set to remotely trigger the jobs are
- USERNAME: This is the username which is used to login on Jenkins server
- PASSWORD: This is the password of the Jenkins user
- JENKINS_URL: This is the URL of the Jenkins server
- Collection data in Build-RoboMQ.postman_collection.json
- Stored API calls for all the jobs Jenkins.postman_environment.json
- Creating a file contains the environment variables that will be needed to run the API calls.

For Jobs that build frontend components

This includes jobs like Dev-Connect-Frontend, Dev-Eventlog-Frontend, Dev-CommunityRecipe-Frontend

- Dev-Connect-Backend: This job is used to build the docker images for connect backend.
- Dev-Eventlog-Frontend: This job is used for building docker images for the event log frontend.
- Dev-Eventlog-Backend: This job is used to build the docker images for event log backend. Location of Jenkinsfile
- Dev-CommunityRecipe-Frontend: This job is used to build docker images for the recipe frontend.
- Dev-Http-Router: This job is used to build the docker images for HTTP Router
- Dev-Deployment-Agent: This job is used to build the docker images for deployment agent. This is a java microservice.
 - It also has a Jenkinsfile which invoked the shell script to create a docker image for deployment agent
 - Arguments what are required for shell script should be in the same order as listed below
 - Tag name – This is the tag with which the docker image being built will be tagged. The default value of tag is “latest”
 - Account name – This is the AWS Account on which we need to push the image. The default value of account is “telesenz”
 - Build User Id – This is the name of the user who will initiate the job.
 - Name of the microservice folder – In this case the name of the microservice folder is Fixed to “FlowDeploymentTransformer”
- **Dev-Event-logger-Microservice**
 - This job is used to build the docker images for Event Logger microservices. It is a combination of two microservices. The trigger (EventLoggerInput) and action (EventLoggerOutput) located in quboid_spring_inputs and quboid_spring_outputs respectively.
 - Arguments what are required for shell script should be in the same order as listed below
 - Tag name – This is the tag with which the docker image being built will be tagged. The default value of tag is “latest”
 - Account name – This is the AWS Account on which we need to push the image. The default value of account is “telesenz”
 - Build User Id – This is the name of the user who will initiate the job.
 - Name of the input microservice folder – This is the path of the input microservice folder. This is set to “quboid_spring_inputs/EventLoggerInput”
 - Name of the output microservice folder – This is the path of the output microservice folder. This is set to “quboid_spring_outputs/EventLoggerOutput”
 - Jenkins job name – This is the name of the Jenkins job. We get this value from the environment variable in jenkins file.

9. Future Enhancements

For Future, our focus is to make the application faster in terms of performance. We will make this search retrieval part as fast as possible. We are also trying to make independent deployments, so that less time is spent on CI/CD.

Now we are using s3 to store data and query it, we will also use object storage or glacier to increase the performance of query. We are also trying to upgrade our existing technology stacks to the latest versions that are available in the market.

10. References

- [1]docs.robomq.io
- [2]docs.docker.com
- [3]kubernetes.io/docs/concepts/
- [4]jenkins.io/doc/



Headquarters
8260 Greensboro Dr.,
Ste A-32, McLean, VA 22102
USA

GST No. **08AAICR2941J1Z4**
Mob no. +91-9829013033
Email: Mohanlal.goel@robomq.io
moanlalgoel@gmail.com

RoboMQ Pvt. Ltd.

India Center
307 Sankalp Tower, 8 Queens Rd,
Jaipur, Rajasthan 302021
India

To Whom-So-Ever It May Concern

Respected Ma'am/Sir,

This is to certify that Aishwaryaditya Jha of JK Lakshmipat University has completed his internship from January 3rd 2022 to June 23rd 2022 under the supervision of our product leads on our product **RoboMQ- Connect iPaaS and Hire2Retire.**

He has completed his training successfully and his performance was good.
We wish him a bright future ahead.

DocuSigned by:

Prachi Sharma

335A8C28C90444A

Date: 7/7/2022

Name: Prachi Sharma

Prachi Sharma
HR Manager and Executive Assistant
Prachi.sharma@robomq.io
+91-8000844735