

Kubernetes Analytics Report



Date: 07/10/2023

Author: Andy Lam

Contents

1.How to create new application.	3
1.1 Base configuration.....	3
1.2 Other configuration	3
2.How to add ingress, reverse ingress.	6
2.1 Ingress	6
2.2 Ingress Controller.....	7
2.3 Add Ingress.....	7
2.4 Reverse Ingress	9
2.5 Add Ingress using manifests file.	9
3.How to change branch from automation.	16
4.Explain the process.....	16
4.1 Define triggering event.	16
4.2 Define environmental variables.	17
4.3 Define jobs.	17
5.How to restart selenium grid if its stuck.	19
5.1 Helm Upgrade	19
5.2 Uninstall helm and reinstall.	19
6.Selenium grid documentation.....	20
7.How to add a new worker to k8s cluster.	23
7.1 Prerequisites to add new worker nodes.....	23
7.2 Join the worker nodes.....	24
8.How to set replica of Maria-db to 2 and sync other.	26
8.1 Backup data.	26
8.2 Secret.....	27
8.3 Service	28
8.4 StatefulSet	29
8.5 Deploy, restore databases and scale up & down.	31
9.How to restart Kubernetes cluster.	32

1.How to create new application.

From the menu select **Applications** then click “+ Add with form” button.

Complete the required information, using the sections below as a guide.

1.1 Base configuration

Field/Option	Overview
Namespace	Select the namespace where the application will reside.
Name	Give the application a descriptive name.
Registry	Select the registry to pull the image from. If you want to pull from a registry that is not configured with Portainer, click Advanced mode then enter the URL and image manually.
Image	Enter the name (and optionally the tag) of the image that will be used to deploy the application.
Annotations	You can add annotations to your application as required by clicking Add annotation and filling in the Key and Value fields.
Stack	Portainer can automatically bundle multiple applications inside a stack. You can either enter the name of a new stack, select an existing stack from the list, or leave empty to use the application name.

Namespace

Namespace

default

Application

Name*

application

Registry

Docker Hub (anonymous)

Image*

docker.io

nginx:latest

Search

Advanced mode

You are currently using an anonymous account to pull images from DockerHub and will be limited to 100 pulls every 6 hours. You can configure DockerHub authentication in the [Registries View](#). Remaining pulls: 99/100

Stack

Portainer can automatically bundle multiple applications inside a stack. Enter a name of a new stack or select an existing stack in the list. Leave empty to use the application name.

Stack


myStack

1.2 Other configuration

- Environment variables

Here you can define any environment variables you wish to be available to your application.

Environment variables + add environment variable


name*	foo	value	bar	
-------	-----	-------	-----	---

- Configuration

Select any Configuration you have previously created to make them available to the application. Portainer will automatically expose all the keys of a Configuration as environment variables. This behavior can be overridden to filesystem mounts for each key via the **Override** button.

Configurations + add configuration

Portainer will automatically expose all the keys of a configuration as environment variables. This behavior can be overridden to filesystem mounts for each key via the override button.

Configuration kube-root-ca.crt Override 

The following keys will be loaded from the kube-root-ca.crt configuration as environment variables: ca.crt

- Persisting data

Define any persisting data within the application and whether these are new or existing volumes, as well as the size of the volume and storage location. To do that, you have to enable a storage option.

You can also define the **Data access policy** for your persisting data:

- **Isolated**: Each instance of the application will use its own data.
- **Shared**: All application instances will use the same data.

- Resource reservations

In this section you can define the amount of **memory** and **CPU** available to the application. If the namespace you have selected has resource quotas set, you must define these values.

Resource reservations

Resource reservations are applied per instance of the application.

Memory limit (MB) ? 10240 94021 10240

CPU limit ? unlimited 4 25.75

- Deployment

This section allows you to choose how you want to deploy the application inside the cluster. Options are:

- **Replicated:** Run one or multiple instances of this container.
- **Global:** Deploy an instance of this container on each cluster node.

You can also define the number of instances of the application to run by setting the **Instance count**.

Deployment

Select how you want to deploy your application inside the cluster.



Replicated

Run one or multiple instances of this container



Global

Application will be deployed as a DaemonSet with an instance on each node of the sdh

Instance
count

3

- Auto-scaling

Toggle **Enable auto scaling for this application** to enable auto-scaling for the application you are deploying. This requires that the Kubernetes metrics server is installed and enabled in the cluster setup.

Field/Option	Overview
Minimum instances	Enter the minimum number of instances that you want running for this application.
Maximum instances	Enter the maximum number of instances that you want running for this application.
Target CPU usage	Enter the target CPU percentage for your application. The autoscaler will ensure that enough instances are running to maintain an average CPU usage of this value across all instances.

Auto-scaling

Enable auto scaling for this application



Minimum instances

1

Maximum instances

1

Target CPU usage (%) ⓘ

50

- Placement preferences and constraints

Here you can define which placement rules must be followed by the nodes where the application is deployed to. Placement rules are based on node labels. To create a new rule, click **add rule**.

You can also define the placement policy for the rules you have set. Options are:

- **Mandatory:** The application will only be scheduled on nodes that follow all rules.
- **Preferred:** If possible, the application will be scheduled on nodes that follow all rules.

Placement preferences and constraints

Placement rules [+ add rule](#)

Deploy this application on nodes that respect **ALL** of the following placement rules. Placement rules are based on node labels.

beta.kubernetes.io/arch

amd64

Placement policy

Specify the policy associated to the placement rules.

Mandatory

Schedule this application **ONLY** on nodes that match **ALL** Rules

Preferred

Schedule this application on nodes that match the rules if possible

- Publishing the application

Here you can create the necessary services to expose your application. Select the type of service (ClusterIP, NodePort, or LoadBalancer) from the dropdown and click **Create service**. You can then configure each service as required.

Publishing the application

Publish your application by creating a ClusterIP service for it, which you may then expose via an ingress.

ClusterIP

+ Create service

ClusterIP

Published ports [+ publish a new port](#)

Container port*

80

Service port*

80

TCP

UDP

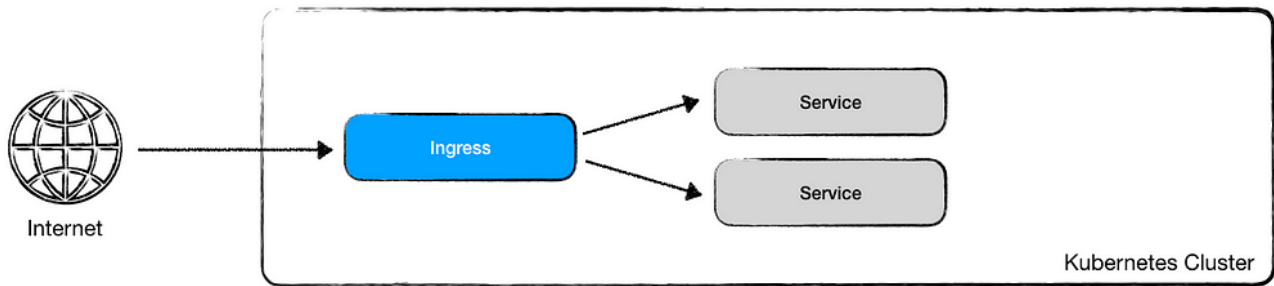
Remove

When you have finished, click **Deploy application**.

2.How to add ingress, reverse ingress.

2.1 Ingress

Ingress is a Kubernetes resource type, that can be applied just like other resources. Its purpose is to define routing cluster-external requests to cluster-internal services. An Ingress will map URLs (hostname and path) to cluster-internal services.



Unlike other resources, Kubernetes has no built-in support for Ingresses. This might sound strange, considering I described Ingress as a Kubernetes native resource. The Ingress is just a description of how routing should be performed. The actual logic has to be performed by an “Ingress Controller”.

2.2 Ingress Controller

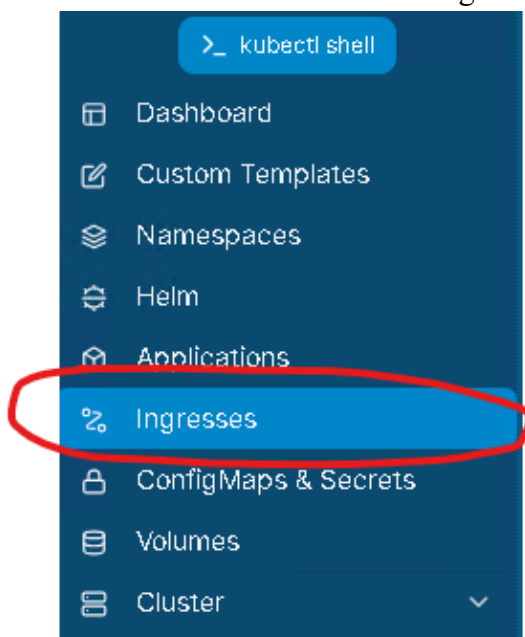
The Ingress is the definition of how the routing should be done. But the execution of those rules has to be performed by an “Ingress Controller”. Due to this, creating Ingress resources in a Kubernetes cluster won’t have any effect until an Ingress Controller is available.

The Ingress Controller is responsible of routing requests to the appropriate services within the Kubernetes cluster. How an Ingress Controller executes this task, is not explicitly defined by Kubernetes. Thus, an Ingress Controller can handle requests in a way that works best for the cluster.

The Ingress Controller is responsible to monitor the Kubernetes cluster for any new Ingress resources. Based on the Ingress resource, the Ingress Controller will setup the required infrastructure to route requests accordingly.

2.3 Add Ingress

1. Log in Portainer.
2. Select Kubernetes Cluster and Ingresses Option.



3. Click “+ Add with form” button.

The screenshot shows the 'Ingresses' management interface. At the top, there's a search bar and three buttons: 'Remove', '+ Add with form' (circled in red), and '+ Create from manifest'. Below is a table listing existing ingress resources. The table has columns for Name, Namespace, Filter, Class Name, Type, and Rules and Paths. The table contains seven rows of ingress resources. At the bottom right, there's a 'Items per page' dropdown set to 10.

<input type="checkbox"/>	Name ↓↑	Namespace ↓↑	Filter	Class Name ↓↑	Type ↓↑	Rules and Paths
<input type="checkbox"/>	corporate-site	prod		public	k8s.io/ingress-nginx	https://corporate.candidatis.io/ → corporate-site:80
<input type="checkbox"/>	database-rest	prod		public	k8s.io/ingress-nginx	https://api.candidatis.io/ → database-rest:80
<input type="checkbox"/>	redis-insight	prod		public	k8s.io/ingress-nginx	https://redis.candidatis.io/ → redis-insight:80
<input type="checkbox"/>	selenium-ingress	default		public	k8s.io/ingress-nginx	http://selenium-grid.local/ → selenium-hub:4444
<input type="checkbox"/>	tigerone	prod		public	k8s.io/ingress-nginx	http://t1.crawler.candidatis.io/ → tigerone:80
<input type="checkbox"/>	tigertwo	prod		public	k8s.io/ingress-nginx	http://t2.crawler.candidatis.io/ → tigertwo:80

4. Select Namespace. In this case, select default.

5. Input “Ingress Name” and select “Ingress class”. In this case, select public as an Ingress class.

6. Fill in the Rule form. (Input Hostname, select TLS secret, service in Kubernetes, Service port, Path type and input Path.) If you want to add another path, click “+ Add Path” button and configure Service, Service port, Path type and Path.

The screenshot shows the 'Ingress' configuration form. It has sections for Namespace, Ingress name, Ingress class, Annotations, Rules, and Paths. Red annotations are present: '4.' points to the 'Namespace*' dropdown (set to 'default'), '5.' points to the 'Ingress class*' dropdown (set to 'public'), and '6.' points to the 'Rule' section. The 'Rule' section contains fields for 'Hostname*' (example.com), 'TLS secret' (selected), and 'No TLS' (deselectable). Below the 'Rule' section is the 'Paths' section with fields for 'Service*' (kubernetes), 'Service port*' (44), 'Path type*' (Exact), and 'Path*' (/). There are also buttons for '+ add annotation', '+ Add rewrite annotation', '+ Add regular expression annotation', '+ Add path', '+ Add new host', and '+ Add fallback rule'.

Namespace* default 4.

Ingress name* example-ingress

Ingress class* public 5.

Annotations

You can specify [annotations](#) for the object. See further Kubernetes documentation on [well-known annotations](#).

+ add annotation + Add rewrite annotation + Add regular expression annotation

Rules

Rule 6.

Hostname* example.com TLS secret No TLS

Add a secret via [ConfigMaps & Secrets](#), then select 'Reload TLS secrets' above to populate the dropdown with your changes.

Paths

Service* kubernetes Service port* 44 Path type* Exact Path* /

+ Add path

+ Add new host + Add fallback rule ?

Notice: In Kubernetes Ingress, there are two commonly used path matching options: "exact" and "prefix." These options define how requests to specific paths are matched and routed within the Ingress controller.

Exact Match: With an exact match, the Ingress controller matches requests only if the path specified in the Ingress resource exactly matches the requested URL path. For example, if you have an Ingress rule with the path /app, it will match requests to /app but not to /app/ or /app/subpath.

Prefix Match: On the other hand, a prefix match allows the Ingress controller to match requests based on a prefix of the requested URL path. If you specify /app/ as the path in the Ingress resource, it will match requests to /app, /app/, and any subpaths such as /app/subpath.

2.4 Reverse Ingress

1. Repeat steps 1-5 of the upper process. (Or select the Ingress you wish to change.)
2. Input Hostname as crawler.candidatis.io.
3. Click "+ Add path" button.
4. Input first Path as t1 and second Path as t2.
5. Click "Create" (Or "Update") button and you can see changed Ingress.

The screenshot displays the Ingress configuration interface. At the top, the 'Namespace' is set to 'prod' and the 'Ingress name' is 'tigerone'. The 'Ingress class' is set to 'public'. Below this, the 'Annotations' section shows four key-value pairs: 'cert-manager.io/cluster-issuer' with value 'letsencrypt-cluster-issuer', 'nginx.ingress.kubernetes.io/auth-realm' with value 'Enter your credentials', 'nginx.ingress.kubernetes.io/auth-secret' with value 'htpasswd', and 'nginx.ingress.kubernetes.io/auth-type' with value 'basic'. There are buttons to '+ add annotation', '+ Add rewrite annotation', and '+ Add regular expression annotation'. The 'Rules' section shows a single rule with 'Hostname' 'crawler.candidatis.io' and 'TLS secret' 'No TLS'. Below the rule, there are two 'Paths' listed: the first with 'Service' 'tigerone', 'Service port' '80', 'Path type' 'Prefix', and 'Path' '/t1'; the second with 'Service' 'tigerone', 'Service port' '80', 'Path type' 'Prefix', and 'Path' '/t2'. A red circle highlights the '+ Add path' button at the bottom left of the paths section.

2.5 Add Ingress using manifests file.

- Deployment Object

```

kind: Deployment
apiVersion: apps/v1
metadata:
  namespace: prod
  name: portal
  labels:
    app: portal
spec:
  replicas: 1
  selector:
    matchLabels:
      app: portal
  template:
    metadata:
      labels:
        app: portal
    spec:
      containers:
        - name: portal
          image: candidatiss/portal-krk-prod
          imagePullPolicy: Always # this is required to force pull image
          ports:
            - containerPort: 8000
          envFrom:
            - configMapRef:
                name: portal-env

```

This code represents a Kubernetes Deployment configuration for an application called “portal” in the “prod” namespace. It specifies that one replica of the application should be created and deployed. The application container uses the image “candidatiss/portal-krk-prod” from Docker Hub and listens on port 8000. It also fetches environment variables from a ConfigMap named “portal-env”.

Below is detailed description of this code.

1. **kind: Deployment:** This line specifies that we're defining a Kubernetes Deployment object. Deployments are used to manage the lifecycle of ReplicaSets, which in turn manage the creation and scaling of Pods.
2. **apiVersion: apps/v1:** It indicates the API version of the object being defined. In this case, it's using the "apps/v1" version of the Kubernetes API.
3. **metadata:** This section contains metadata information about the Deployment object, such as its name, namespace, and labels.
 - **namespace: prod:** The Deployment is created within the "prod" namespace. Namespaces provide a way to logically divide cluster resources.
 - **name: portal:** The name assigned to the Deployment object is "portal".

- labels: A set of key-value pairs used to attach identifying metadata to the Deployment object. In this case, the label "app: portal" is specified.

4. spec: The spec section defines the desired state for the Deployment object.

- replicas: 1: Specifies that one replica of the Deployment should be created. A replica is an instance of the application running as a Pod.

- selector: Defines the criteria used to select which Pods are managed by this Deployment.

- matchLabels: Specifies that the Pods must have the label "app: portal" to be selected.

- template: Describes the Pod template used to create new Pods when necessary.

- metadata: Contains metadata specific to the Pod template.

- labels: Assigns the label "app: portal" to the Pods created from this template.

- spec: Specifies the specification of the Pod template.

- containers: Defines the list of containers to run within the Pod.

- - name: portal: Specifies the name of the container as "portal".

- image: candidatis/portal-krk-prod: Sets the Docker image to be used for the container. In this case, it's pulling the image named "candidatis/portal-krk-prod".

- imagePullPolicy: Always: Specifies that the container image should always be pulled from the registry, even if it already exists locally.

- ports: Defines the ports that should be opened in the container.

- - containerPort: 8000: Specifies that port 8000 within the container should be accessible.

- envFrom: Specifies configuration data to be provided to the container from a ConfigMap.

- - configMapRef: Specifies that the data should be obtained from a ConfigMap resource.

- name: portal-env: The ConfigMap named "portal-env" is referenced, providing environment variables to the container.

- **Service Object**

```
---  
kind: Service
```

```
apiVersion: v1
metadata:
  namespace: prod
  name: portal
spec:
  selector:
    app: portal
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
```

This code represents a Kubernetes Service configuration for the "portal" application in the "prod" namespace. The Service acts as an internal load balancer for the Deployment, allowing other components within the cluster to access the "portal" application. It uses the ClusterIP type, which provides a stable IP address that is only accessible within the cluster, and it maps incoming traffic on port 80 to the target port 8000 of the pods selected by the label selector "app: portal".

Below is detailed description of this code.

1. kind: Service: This is a YAML key-value pair that specifies the kind of Kubernetes resource being defined, which in this case is a Service object.
2. apiVersion: v1: This indicates the API version of Kubernetes being used. In this case, it's version 1.
3. metadata:: This YAML key indicates the start of the metadata section, which provides additional information about the Service object.
 - namespace: prod: The namespace field specifies the namespace in which the Service object will be created. Namespaces are a way to organize and isolate resources within a Kubernetes cluster. In this case, the Service will be created in the "prod" namespace.
 - name: portal: The name field specifies the name of the Service object. In this case, the Service will be named "portal".
4. spec:: This YAML key indicates the start of the specification section, which defines the desired state of the Service object.
 - selector:: The selector field is used to select the Pods on which the Service will operate. In this case, the selector is set to app: portal, which means the Service will target Pods with the label app set to "portal".
 - type: ClusterIP: The type field specifies the type of Service. In this case, it is set to ClusterIP, which exposes the Service on an internal IP address that is only accessible within the cluster.

- ports:: The ports field specifies the ports that the Service will listen on and forward traffic to the targeted Pods.

- - protocol: TCP: This line starts the definition of a port within the ports array. Here, the protocol field is set to TCP, indicating that TCP traffic will be forwarded.

- port: 80: The port field specifies the port on which the Service will listen for incoming traffic. In this case, the Service will listen on port 80.

- targetPort: 8000: The targetPort field specifies the port to which the incoming traffic will be forwarded within the targeted Pods. Here, the traffic will be forwarded to port 8000 of the Pods.

- Ingress Object

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: portal
  namespace: prod
  annotations:
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: httpasswd
    nginx.ingress.kubernetes.io/auth-realm: "Enter your credentials"
    cert-manager.io/cluster-issuer: "letsencrypt-cluster-issuer"
spec:
  tls:
  - hosts:
    - jusjobs.candidatis.io
    - tecjobs.candidatis.io
    secretName: cluster-certificate-secret
  rules:
  - host: jusjobs.candidatis.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: portal
            port:
              number: 80
  - host: tecjobs.candidatis.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
```

```
name: portal
port:
  number: 80
```

This code represents a Kubernetes Ingress configuration for the "portal" application in the "prod" namespace. It defines rules for routing external traffic to the application based on the requested hostnames. The Ingress also includes annotations for enabling basic authentication using an httpasswd secret and specifies a TLS certificate issued by "letsencrypt-cluster-issuer" for the hosts "jusjobs.candidatis.io" and "tecjobs.candidatis.io".

1. apiVersion: networking.k8s.io/v1: This line specifies the API version of Kubernetes Ingress being used, which is networking.k8s.io/v1.

2. kind: Ingress: This indicates that we are defining an Ingress resource in Kubernetes. Ingress is an API object that manages external access to services within a cluster.

3. metadata:: This YAML key indicates the start of the metadata section, providing additional information about the Ingress object.

- name: portal: The name field specifies the name of the Ingress object, which is "portal" in this case.

- namespace: prod: The namespace field defines the namespace in which the Ingress object will be created. In this case, the Ingress will be created in the "prod" namespace.

- annotations:: The annotations field allows you to add custom key-value pairs to the Ingress object for additional configuration or information.

- nginx.ingress.kubernetes.io/auth-type: basic: This annotation specifies the authentication type as "basic". It indicates that the Ingress should use basic authentication for incoming requests.

- nginx.ingress.kubernetes.io/auth-secret: httpasswd: This annotation specifies the secret name where the credentials for basic authentication are stored. In this case, the secret is named "httpasswd".

- nginx.ingress.kubernetes.io/auth-realm: "Enter your credentials": This annotation sets the authentication realm for basic authentication. The value "Enter your credentials" will be displayed to users when prompted for credentials.

- cert-manager.io/cluster-issuer: "letsencrypt-cluster-issuer": This annotation specifies the cluster issuer for TLS (Transport Layer Security) certificates. It indicates that the Ingress should use the "letsencrypt-cluster-issuer" to obtain TLS certificates.

4. spec:: This YAML key indicates the start of the specification section, which defines the desired state of the Ingress object.

- tls:: The tls field specifies the TLS configuration for the Ingress, enabling secure HTTPS connections.
- - hosts:: This line starts the definition of hosts for which the Ingress will handle traffic. In this case, two hosts are specified: "jusjobs.candidatis.io" and "tecjobs.candidatis.io".
 - jusjobs.candidatis.io: This line specifies the first host for which the Ingress will handle traffic.
 - tecjobs.candidatis.io: This line specifies the second host for which the Ingress will handle traffic.
- secretName: cluster-certificate-secret: This line specifies the name of the secret that contains the TLS certificate and private key for the specified hosts.
- rules:: The rules field defines the routing rules for incoming requests based on the specified hosts.
 - - host: jusjobs.candidatis.io: This line specifies the host for which the following routing rules will be applied.
 - http:: This line indicates that the routing rules will handle HTTP traffic.
 - paths:: The paths field specifies the URL paths for which the routing rules will be applied.
 - path: /: This line specifies the URL path "/" for which the following routing rules will be applied.
 - pathType: Prefix: This line specifies that the matching rule for the path is a prefix match. Any request starting with "/" will match this rule.
 - backend:: The backend field specifies the backend service to which the matching requests will be forwarded.
 - service:: This line indicates that the backend is a Kubernetes Service.
 - name: portal: This specifies the name of the Service to which the requests will be forwarded, which is "portal" in this case.
 - port:: The port field specifies the port on which the Service is listening.
 - number: 80: This line specifies that the requests should be forwarded to port 80 of the Service.
 - - host: tecjobs.candidatis.io: This line specifies the second host for which the routing rules will be applied, following a similar structure as explained above.

- Deploy via CLI

After creating the manifest file, execute the following command. (in this case, name of manifest file is manifest.yaml)

```
kubectl apply -f manifest.yaml
```

After executing this command, you can check status by using the following command.

```
kubectl get all -A
```

Or you can only check the status of a specific namespace by using following command.

```
kubectl get all -n <namespace>
```

3.How to change branch from automation.

```
on:
  push:
    branches: [ main,miw-prod,olw-prod,flb-prod,ald-prod,izb-prod,flh-prod,krk-prod ]
```

There is .github/workflow/workflow.yml file in source code. In this file, you can see above code. In this case, code is defining a rule for the “push” event. Whenever a push event occurs in the repository, the specified branches will be considered for triggering any associated actions or workflows. The branches specified in the code are: main, mix-prod, olw-prod, flb-prod, ald-prod, izb-prod, flh-prod, krk-prod. If you want to change branch from automation, you can add or remove branch name from “branches” list.

4.Explain the process.

The code is a GitHub Actions workflow that consists of two jobs: “test” and “build”. This is CI (Continuous Integration) workflow, not CD (Continuous Delivery and Continuous Deployment). This workflow sets up a Python environment, installs dependencies, runs tests and then builds and pushes a Docker image to Docker hub. Below is a detailed description of code.

4.1 Define triggering event.

```
on:
  push:
    branches: [ main,miw-prod,olw-prod,flb-prod,ald-prod,izb-prod,flh-prod,krk-prod ]
```

As I said, this code is defining a rule for the “push” event. A description of this is omitted.

4.2 Define environmental variables.

```
env:  
  BRANCH_NAME: ${github.head_ref || github.ref_name }
```

This code sets the value of the environmental variable “BRANCH_NAME” based on the value of either “github.head_ref” or “github.ref_name”.

4.3 Define jobs.

```
jobs:  
  test:  
    runs-on: ubuntu-latest  
    strategy:  
      max-parallel: 4  
      matrix:  
        python-version: [ 3.11 ]  
    steps:  
      - uses: actions/checkout@v3  
      - name: Setup Python  
        uses: actions/setup-python@v3.0.0  
        with:  
          # Version range or exact version of a Python version to use, using  
          # SemVer's version range syntax.  
          python-version: 3.11.1  
          # Used to specify a package manager for caching in the default directory.  
          # Supported values: pip, pipenv.  
          cache: pip  
      - name: Install Dependencies  
        run: |  
          python -m pip install --upgrade pip  
          pip install -r requirements.txt  
      - name: Run Tests  
        run: |  
          python manage.py test  
  
  build:  
    runs-on: ubuntu-latest  
    needs: test  
    steps:  
      - uses: actions/checkout@v3  
        name: Check out code  
      - uses: mr-smithers-excellent/docker-build-push@v5  
        name: Build & push Docker image  
        with:  
          image: candidatis/portal-${BRANCH_NAME}  
          tags: latest  
          registry: docker.io  
          username: ${secrets.DOCKER_USERNAME}  
          password: ${secrets.DOCKER_PASSWORD}
```

This code sets up two jobs. The “test” job sets up the Python environment, installs dependencies, and runs tests. The “build” job depends on the completion of the “test” job and builds and pushes a Docker image using the code checked out from the repository.

```
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      max-parallel: 4
      matrix:
        python-version: [ 3.11 ]
    steps:
```

- This code defines a job named "test" that runs on the latest version of Ubuntu.
- The "strategy" section specifies that a maximum of 4 parallel jobs can run.
- The "matrix" section defines a matrix variable named "python-version" with a single value of 3.11.
- The "steps" section contains the list of steps to be executed within this job.

```
- uses: actions/checkout@v3
```

- This step uses the actions/checkout@v3 action to fetch the repository's code.

```
  uses: actions/setup-python@v3.0.0
  with:
    python-version: 3.11.1
    cache: pip
```

- This step uses the actions/setup-python@v3.0.0 action to set up the Python environment.
- It specifies the Python version as 3.11.1 and uses pip as the package manager for caching.

```
- name: Install Dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

- This step installs the necessary dependencies by upgrading pip and then installing the packages specified in the requirements.txt file.

```
- name: Run Tests
  run: |
    python manage.py test
```

- This step runs the tests using the command "python manage.py test".

```
build:
  runs-on: ubuntu-latest
  needs: test
  steps:
```

- This code defines another job named "build" that also runs on the latest version of Ubuntu.
- The "needs" keyword specifies that this job depends on the completion of the "test" job.

```
- uses: actions/checkout@v3
  name: Check out code
```

- This step uses the actions/checkout@v3 action to fetch the repository's code.

```
- uses: mr-smithers-excellent/docker-build-push@v5
  name: Build & push Docker image
  with:
    image: candidatis/portal-${BRANCH_NAME}
    tags: latest
    registry: docker.io
    username: ${ secrets.DOCKER_USERNAME }
    password: ${ secrets.DOCKER_PASSWORD }
```

- This step uses the mr-smithers-excellent/docker-build-push@v5 action to build and push a Docker image.
- It specifies the image name as "candidatis/portal-\${BRANCH_NAME}" where \${BRANCH_NAME} will be replaced with the branch name being built.
- The image is tagged as "latest" and pushed to the docker.io registry using the provided username and password stored as secrets in GitHub.

5.How to restart selenium grid if its stuck.

Selenium-grid is deployed using helm chart, so you can restart selenium-grid by using helm

There are 2 ways to restart selenium-grid.

5.1 Helm Upgrade

- Find Helm Chart source code.

```
cd ~/selenium|
```

- Restart selenium grid pods by using following command-line.

```
helm upgrade --recreate-pods selenium-grid selenium-grid/|
```

- After executing this command, you can see the pod restart.

5.2 Uninstall helm and reinstall.

- List helm by using following command-line. You can see that selenium-grid is installed by using helm.

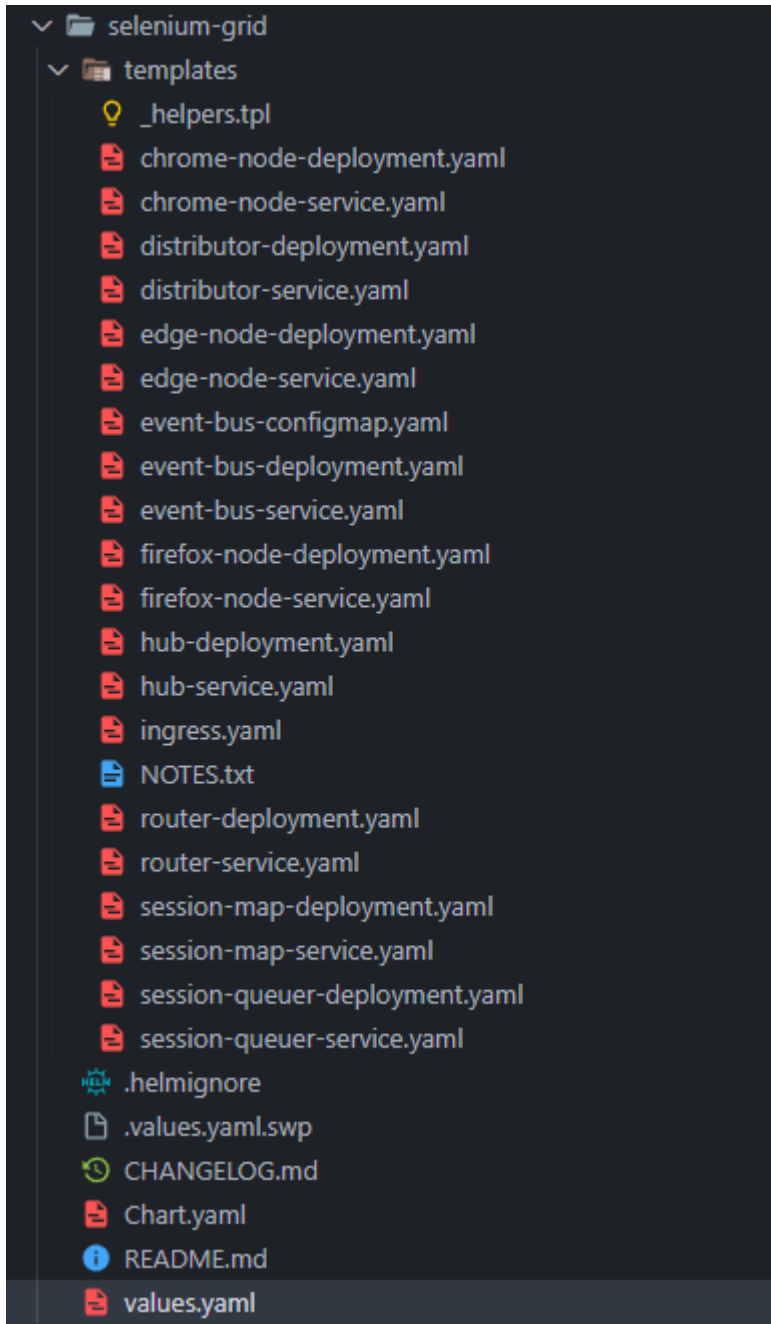
```
> helm list
NAME          NAMESPACE    REVISION    UPDATED                               STATUS    CHART          APP VERSION
selenium-grid default        1           2023-06-05 12:36:47.843198455 +0000 UTC deployed selenium-grid-0.15.7 4.8.3-20230403
```

- Uninstall and reinstall selenium-grid.

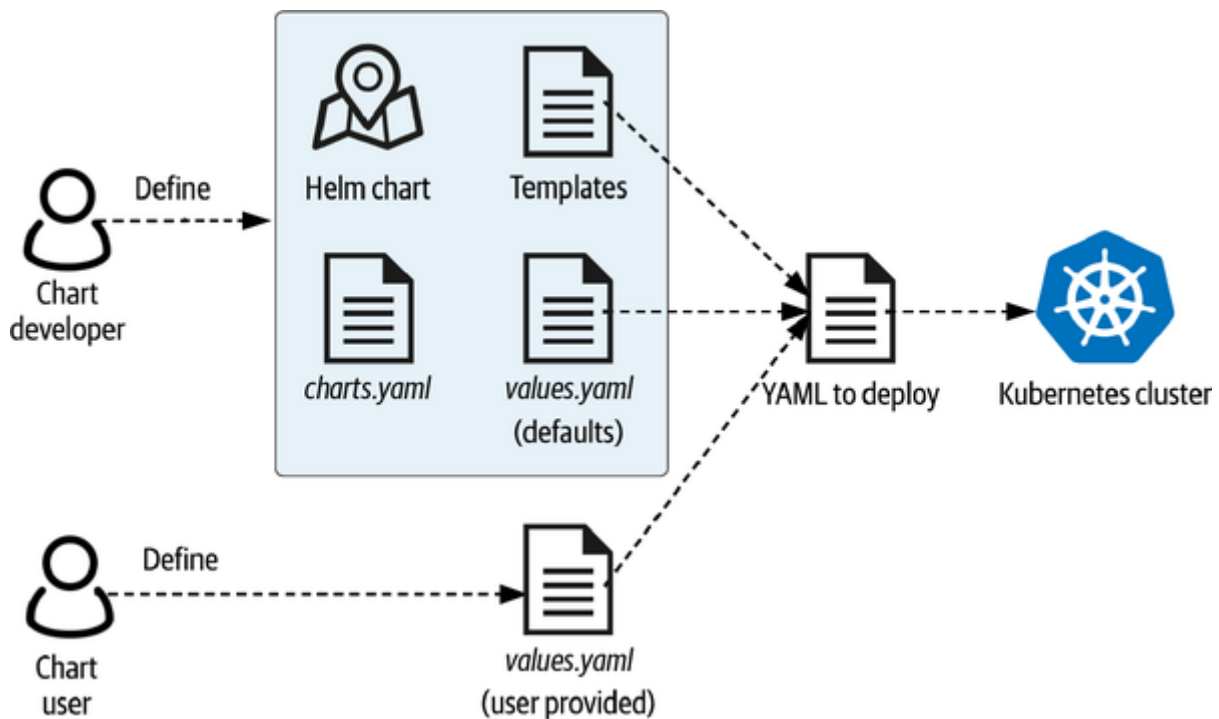
```
> helm delete selenium-grid && helm install <selenium-grid helm chart folder>
```

6.Selenium grid documentation.

Below is file structure of Selenium-grid.



As you can see, Helm is consisted of README.md, values.yaml, Chart.yaml, .helmignore and templates directory.



First, looking at the contents of Helm chart.

- **README file**

This explains how to use the chart. These instructions are provided along with the chart in registries.

- **Chart.yaml file.**

This contains metadata about the chart such as its name, publisher, version, keywords, and any dependencies on other charts. These properties are useful when searching Helm registries to find charts.

- **values.yaml file**

This lists out the configurable values supported by the chart and their default values. These files typically contain a good number of comments that explain the available options.

- **templates directory**

This contains **Go templates** that define the chart. The templates include a Notes.txt file used to generate the output you saw previously after executing the **helm install** command, and one or more YAML files that describe a pattern for a Kubernetes resource. These YAML files may be organized in subdirectories (for example, the template that defined a StatefulSet for MariaDB primary replicas). Finally, a **_helpers.tpl** file describes how to use the templates. Some of the templates may be used multiple times or not at all, depending on the selected configuration values.

This figure shows how helm works. When you executed the **helm install** command, the Helm client makes sure it has an up-to-date copy of the chart you have named by checking with the source repository. Then it uses the template to generate YAML configuration code, overriding default values from the chart's **values.yaml** file with any values you have provided. It then uses **kubectl** command to apply this configuration to your currently configured Kubernetes clusters.

If you would like to see the configuration that Helm chart will produce before applying it, you can use the handy **template** command.

```
helm template selenium-grid selenium-grid
```

After executing this command, you can see the Kubernetes configuration result.

```
---
# Source: selenium-grid/templates/event-bus-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: selenium-event-bus-config
  namespace: default
  labels:
    app.kubernetes.io/managed-by: helm
    app.kubernetes.io/instance: selenium-grid
    app.kubernetes.io/version: 4.8.3-20230403
    app.kubernetes.io/component: selenium-grid-4.8.3-20230403
    helm.sh/chart: selenium-grid-0.15.7
data:
  SE_EVENT_BUS_HOST: selenium-hub
  SE_EVENT_BUS_PUBLISH_PORT: "4442"
  SE_EVENT_BUS_SUBSCRIBE_PORT: "4443"
---
# Source: selenium-grid/templates/chrome-node-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: selenium-chrome-node
  namespace: default
  labels:
    name: selenium-chrome-node
    app.kubernetes.io/managed-by: helm
```

After verifying the result is correct or not, you can install selenium-grid using the helm install command.

```
helm install selenium-grid selenium-grid
```

If selenium-grid is installed successfully after executing this command, you can see the result as below.

```
NAME: selenium-grid
LAST DEPLOYED: Sat Jul 8 14:42:19 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Selenium Grid Server deployed successfully.
1. Ingress is enabled, and it exposes the Grid Hub or Grid Router with the hostname you supplied.
   To access Selenium from outside of Kubernetes, simply open http://selenium-grid.local.
2. Within Kubernetes cluster, you can use following Service endpoint:
   http://selenium-hub.default.svc:4444
```

You can see that selenium-grid is successfully installed on Kubernetes by using **kubectl** and **helm list** command.

```

pod/selenium-chrome-node-5f44bffc9b-bgprk 1/1 Running 0 5m5s
pod/selenium-edge-node-795fdb8484-75lsn 1/1 Running 0 5m5s
pod/selenium-firefox-node-6496647db-rddcm 1/1 Running 0 5m5s
pod/selenium-hub-7c4785d969-z8x4j 1/1 Running 0 5m5s
service/selenium-chrome-node ClusterIP 10.102.82.123 <none> 6900/TCP 5m5s
service/selenium-edge-node ClusterIP 10.109.60.92 <none> 6900/TCP 5m5s
service/selenium-firefox-node ClusterIP 10.100.254.107 <none> 6900/TCP 5m5s
service/selenium-hub ClusterIP 10.105.216.119 <none> 4444/TCP, 4442/TCP, 4443/TCP 5m5s
deployment.apps/selenium-chrome-node 1/1 1 1 5m5s
deployment.apps/selenium-edge-node 1/1 1 1 5m5s
deployment.apps/selenium-firefox-node 1/1 1 1 5m5s
deployment.apps/selenium-hub 1/1 1 1 5m5s
replicaset.apps/selenium-chrome-node-5f44bffc9b 1 1 1 5m5s
replicaset.apps/selenium-edge-node-795fdb8484 1 1 1 5m5s
replicaset.apps/selenium-firefox-node-6496647db 1 1 1 5m5s
replicaset.apps/selenium-hub-7c4785d969 1 1 1 5m5s

```

```

> helm list
NAME                NAMESPACE      REVISION      UPDATED                                 STATUS      CHART
selenium-grid       default         1             2023-06-05 12:36:47.843198455 +0000 UTC deployed selenium-grid-0.15.7

```

As you can see, there are 4 pods: selenium-chrome-node, selenium-edge-node, selenium-firefox-node and selenium-hub. But in your Kubernetes cluster, you can see only selenium-chrome-node and selenium-hub. Because previous engineer deleted selenium-firefox-node and selenium-edge-node. You can delete resource by using **kubectl delete** command. You have to delete both service and deployment.

```

kubectl delete deployment/selenium-edge-node
kubectl delete service/selenium-edge-node

```

You can delete entire selenium-grid by using **helm uninstall** command.

```

helm uninstall selenium-grid

```

And in your Kubernetes cluster, a number of selenium-chrome-node pods is 60. (Default, replica of selenium-chrome-node is 1.) You can scale up & down by using **kubectl scale** command.

```

kubectl scale deployment.apps/selenium-chrome-node --replicas=60

```

7.How to add a new worker to k8s cluster.

7.1 Prerequisites to add new worker nodes

1. A new node with minimum 2CPU's with 4Gb memory is required. Operating system should be installed and ready for the setup.
2. Make sure Kubernetes master and new worker node is reachable between each other.
3. Kubernetes doesn't support "Swap", so disable Swap on new node using below command and also to make it permanent comment out the swap entry in /etc/fstab file.

```

sudo swapoff -a

```

4. Internet must be enabled on new node, because required packages will be downloaded from official repository.

7.2 Join the worker nodes.

1. Get the joining Token first from Kubernetes Master Node.

- Log in to Kubernetes Master node and get the joining token as below.

```
user1@kubernetes-master:~$ kubeadm token list
```

- If no join token is available, generate new join token using kubeadm command.

```
user1@kubernetes-master:~$ kubeadm token create --print-join
-command
W0331 13:10:57.055398 12062 configset.go:202] WARNING: kub
eadm cannot validate component configs for API groups [kubele
t.config.k8s.io kubeproxy.config.k8s.io]
kubeadm join 192.168.2.1:6443 --token jnnu6g.wqbmmc1l2xtdf40
t --discovery-token-ca-cert-hash sha256:ce4c91f6f5442c8c8
519cacd4673864f3ce5e466435a6f6ac9e877d1c831f6dc
user1@kubernetes-master:~$ kubeadm token list
TOKEN          TTL    EXPIRES          USAGES
DESCRIPTION  EXTRA GROUPS
jnnu6g.wqbmmc1l2xtdf40t  23h    2020-04-01T13:10:57+05:3
0 authentication,signing <none>    system:bootstrappers:kub
eadm:default-node-token
```

- Copy the token highlighted in yellow color to join the worker nodes and keep it aside.

2. On Worker nodes, perform the below steps to join the worker nodes.

- Update the Ubuntu Repositories and install basic tools like apt-transport-https, curl.

```
sudo apt-get update && sudo apt-get install -y apt-transport-https curl|
```

- Add the Kubernetes Signing key on new worker nodes.


```
sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

- Add Kubernetes repository on new worker nodes.

```
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
```

- Install the below required packages on new worker nodes using apt-get command

```
sudo apt-get update && sudo apt-get install -y kubelet kubeadm kubectl docker.io
```

- Start and Enable Docker service on new worker nodes.

```
sudo systemctl enable docker
```

- Use the joining token you have copied earlier to add the worker nodes.

```
user1@kubernetes-worker3:~$ sudo kubeadm join 192.168.2.1:6443 --token jnnu6g.wqbmme1l2xtdf40t
--discovery-token-ca-cert-hash sha256:ce4c91f6f5442c8c8519cacd4673864f3ce5e466435a6f6ac9e8
77d1c831f6dc
W0331 14:00:32.775291 7193 join.go:346] [preflight] WARNING: JoinControlPlane.controlPlane settings
will be ignored when control-plane flag is not set.
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroups" as the Docker cgroup driver. The recommend
ed driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oy
aml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.18" ConfigMap in t
he kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:

* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

user1@kubernetes-worker3:~$
```

- Our new worker node is joined to the cluster successfully as per the above output. Lets login to Kubernetes master and confirm the node list.

3. On Kubernetes Master

Verify the list of nodes using kubectl command

```
user1@kubernetes-master:~$ kubectl get nodes
NAME                STATUS  ROLES  AGE  VERSION
kubernetes-master   Ready   master  71m  v1.18.0
kubernetes-worker1  Ready   <none>  69m  v1.18.0
kubernetes-worker2  Ready   <none>  69m  v1.18.0
kubernetes-worker3  Ready   <none>  59m  v1.18.0
```

That's it. Above output shows our newly added worker nodes.

8. How to set replica of Maria-db to 2 and sync other.

While deploying the MariaDB on Kubernetes, using StatefulSet is better than using deployment.

StatefulSet is the Kubernetes object used to manage stateful applications. It is preferred over deployments as it provides guarantees about the ordering and uniqueness of these Pods i.e., the management of volumes is better with stateful sets.

MariaDB is going to be a stateful application i.e., it stores data (like tables, users) inside a volume. If the data is stored in pod ephemeral storage, then the data will get erased once the pod restarts.

Also, MariaDB may have to be scaled to more than one pod in caseload increases.

All these operations have to be done in such a way that data consistency is maintained across all the pods like mariadb-0, mariadb-1.

8.1 Backup data.

To convert a Deployment to a StatefulSet in Kubernetes, we have to backup data. Converting to a StatefulSet involves deleting and recreating pods, which may result in data loss if not properly handled. So, you have to backup data for protecting data loss.

- Identify the MariaDB pods.

Use ***kubectl get pods*** command to identify the pod name of your MariaDB deployment.

- Create a backup directory or storage.

Set up a backup directory or storage where you will store the backup files. This could be a local directory on your machine or a network storage location.

- Connect the MariaDB pod.

Execute an interactive shell session within the MariaDB pod using the following command.

```
kubectl exec -it pods/mariadb-prod-5db4799458-k9hbm -n prod -- /bin/bash
```

-Perform the backup.

Within the pod shell, you can use *mysqldump* command to perform the backup. To backup all databases, you can use following command.

```
root@mariadb-prod-5db4799458-k9hbm:/# mariadb-dump -u root -p --all-databases >> all-databases.sql
```

And enter password, backup is processing. Wait for the command to complete, it depends on the size of databases, it may take some time.

- Copy the backup file.

```
kubectl cp pods/mariadb-prod-5db4799458-k9hbm:/<path>/all-database-sql /<localpath>/all-database.sql
```

8.2 Secret

Secrets in Kubernetes are the objects used for supplying sensitive information to containers. They are like ConfigMaps with the difference that data is store in a base 64 encoded format.

For the Security of MariaDB cluster, it is wise to restrict access to the database with a password. We will use Secrets to mount our desired passwords to the containers.

In this case, we use base64 encoded to store “ROOT_PASSWORD”.

- Get base64 encoded password.

```
> echo "password" | base64
cGFzc3dvcmQK
```

- Create Stateful set manifest file: sts.yaml. (This file is combination of secret, service and statefulset manifest files.)

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  ROOT_PASSWORD: cGFzc3dvcmQ=
```

This code create a Kubernetes Secret named “myscret” with a single key-value pair where the key is “ROOT_PASSWORD” and the value is the base64-encoded representation of the root password.

1. apiVersion: v1

- Specifies the version of the Kubernetes API being used.

2. kind: Secret

- Defines the type of resource being created, which is a Secret in this case.

3. metadata:

name: mysecret

- Specifies the metadata for the Secret resource, including its name.

4. type: Opaque

- Specifies the type of the Secret. "Opaque" means that the secret's data is not base64-encoded.

5. data:

ROOT_PASSWORD: cGFzc3dvcmQ=

- Defines the data section of the Secret, where the key-value pairs are specified. In this case, the key is "ROOT_PASSWORD" and the value is "cGFzc3dvcmQ=". The value appears to be base64-encoded, representing the actual root password.

8.3 Service

```
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb
  namespace: mariadb
  labels:
    app: mariadb
spec:
  ports:
    - port: 3306
      name: mariadb
  selector:
    app: mariadb
    tier: mariadb-cache
  clusterIP: None
```

This code creates a Kubernetes Service object named “mariadb” in the “mariadb” namespace. The Service exposes port 3306 and selects Pods with the labels “app:mariadb” and “tier:mariadb-cache”. The Service doesn’t have a clusterIP, making it only accessible from within the cluster.

1. apiVersion: v1: Specifies the API version of Kubernetes being used.
2. kind: Service: Defines that the Kubernetes object being created is a Service.
3. metadata: Contains metadata information about the Service.
 - name: mariadb: Specifies the name of the Service as "mariadb".
 - namespace: mariadb: Specifies the namespace where the Service will be created as "mariadb".
 - labels: Defines labels for the Service.
4. spec: Specifies the specifications of the Service.
 - ports: Defines the ports to be exposed by the Service.
 - port: 3306: Specifies that port 3306 should be exposed by the Service.
 - name: mariadb: Assigns the name "mariadb" to the port.
 - selector: Defines the labels that the Service will use to select the corresponding Pods.
 - app: mariadb: Selects Pods with the label "app" as "mariadb".

- tier: mariadb-cache: Selects Pods with the label "tier" as "mariadb-cache".
- clusterIP: None: Specifies that the Service should not have a clusterIP. This means that the Service will not be accessible from outside the cluster, and it will only be reachable from within the cluster.

8.4 StatefulSet

```
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mariadb
  namespace: mariadb
spec:
  replicas: 2
  serviceName: mariadb
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: mariadb
          image: mariadb:5.6
          ports:
            - name: tpc
              protocol: TCP
              containerPort: 3306
          env:
            - name: mariadb_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: ROOT_PASSWORD
                  name: mysecret
          volumeMounts:
            - name: data
              mountPath: /var/lib/mysql
      volumeClaimTemplates:
        - metadata:
            name: data
          spec:
            storageClassName: standard
            accessModes:
              - ReadWriteOnce
            resources:
```

```
requests:
  storage: 10Gi
```

This code describes a StatefulSet resource for deploying a MariaDB database with 2 replicas, using a specific image and environment variables sourced from a secret. It also defines a PersistentVolumeClaim template for storage needs.

1. apiVersion: apps/v1: Specifies the API version of Kubernetes being used for the StatefulSet resource.
2. kind: StatefulSet: Defines that the Kubernetes object being created is a StatefulSet.
3. metadata: Contains metadata information about the StatefulSet.
 - name: mariadb: Specifies the name of the StatefulSet as "mariadb".
 - namespace: mariadb: Specifies the namespace of the StatefulSet as "mariadb".
4. spec: Specifies the specifications of the StatefulSet.
 - replicas: 2: Specifies that two replicas of the StatefulSet should be created.
 - serviceName: mariadb: Specifies the name of the service that will be created for accessing the StatefulSet as "mariadb".
 - selector: Defines the labels that will be used to select the pods belonging to this StatefulSet.
 - matchLabels: Specifies the label selector criteria for selecting the pods.
 - app: mariadb: Selects pods with the label "app" as "mariadb".
 - template: Defines the template for creating the pods in the StatefulSet.
 - metadata: Contains metadata information about the pods.
 - labels: Specifies the labels for the pods.
 - app: mariadb: Assigns the label "app" as "mariadb" to the pods.
 - spec: Specifies the specifications for the pods.
 - terminationGracePeriodSeconds: 10: Defines the termination grace period for the pods as 10 seconds.
 - containers: Specifies the containers that will be running in the pods.
 - name: mariadb: Specifies the name of the container as "mariadb".
 - image: mariadb:5.6: Specifies the Docker image to be used for the container.
 - ports: Specifies the ports to be exposed by the container.
 - name: tpc: Assigns the name "tpc" to the port.
 - protocol: TCP: Specifies the protocol for the port as TCP.
 - containerPort: 3306: Specifies that port 3306 should be exposed by the container.
 - env: Specifies environment variables for the container.

- name: mariadb_ROOT_PASSWORD: Specifies the name of the environment variable as "mariadb_ROOT_PASSWORD".
- valueFrom: Specifies that the value of the environment variable will be sourced from another Kubernetes resource.
 - secretKeyRef: Specifies that the value will be sourced from a secret key.
 - key: ROOT_PASSWORD: Specifies the key of the secret from which the value will be sourced.
 - name: mysecret: Specifies the name of the secret from which the value will be sourced.
- volumeMounts: Specifies the volumes that will be mounted inside the container.
 - name: data: Specifies the name of the volume as "data".
 - mountPath: /var/lib/mysql: Specifies the path at which the volume will be mounted inside the container.
- volumeClaimTemplates: Specifies the template for creating PersistentVolumeClaims (PVCs) for the StatefulSet.
 - metadata: Contains metadata information about the PVC template.
 - name: data: Specifies the name of the PVC template as "data".
 - spec: Specifies the specifications for the PVC template.
 - storageClassName: standard: Specifies the storage class to be used for the PVC template as "standard".
 - accessModes: Specifies the access mode for the PVC template.
 - ReadWriteOnce: Specifies that the PVC can be mounted as read-write by a single node.
 - resources: Specifies the resource requirements for the PVC template.
 - requests: Specifies the requested resources for the PVC template.
 - storage: 10Gi: Specifies that the PVC should request 10 gigabytes of storage.

8.5 Deploy, restore databases and scale up & down.

- Deploy

To deploy MariaDB to Kubernetes Cluster, execute following command-line.

```
kubectl apply -f sts.yaml
```

After executing above command-line, you will see MariaDB pods running.

- Restore databases

- Copy data to pods.

Identify the MariaDB pod by using **kubectl get pods** command. If you deployed MariaDB using StatefulSet, you will see Pod name in the format mariadb-0, mariadb-1.

Copy data to mariadb pods by using following command.

```
kubectl cp <localpath>/all-database.sql pods/mariadb-0:<path>/
```

And restore database by using below command.

```
kubectl exec -it pods/mariadb-0 -- ls <path>|  
mysql -u root -p < all-databases.sql|
```

- Scale up & down

To scale up MariaDB Cluster, execute following command-line.

```
kubectl scale sts/mariadb -n mariadb --replicas=3|
```

To scale down MariaDB Cluster, decrease replicas number and execute command line.

```
kubectl scale sts/mariadb -n mariadb --replicas=0|
```

9.How to restart Kubernetes cluster.

This Kubernetes cluster is deployed using Microk8s. Microk8s is a lightweight, minimalistic distribution of Kubernetes that is designed for local development, testing and small-scale production deployments. Microk8s is not intended for large-scale, high-availability, or mission-critical production deployments. If you want to large-scale and high-availability, I recommend you to use kubeadm instead of Microk8s.

To restart Kubernetes cluster, perform the following steps.

- Stop the Microk8s cluster on each node. (If you want, you can restart node.)

```
microk8s stop|
```

- Start the Microk8s cluster on each node.

```
microk8s start|
```

- Verify the status of the cluster on each node.

```
microk8s status --wait-ready
```

This command will wait until the cluster is ready and display the status of the individual components.

- Verify the cluster connectivity by running a basic Kubernetes command, such as checking the nodes:

```
microk8s kubectl get nodes
```

By following these steps, you can restart the Microk8s cluster on each node and ensure that the cluster is up and running again.