

Covid-19 Data Analytics with Kubernetes

Microservices Application with Kubernetes and OpenShift

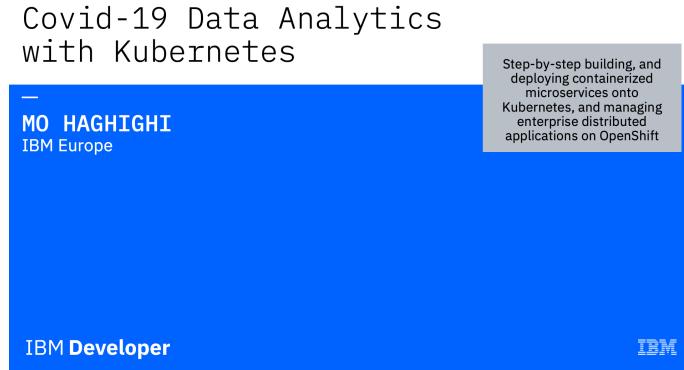
Table of contents

1. Covid-19 Data Analytic Microservices Application with Kubernetes and OpenShift	7
1.1 Watch the full series on YouTube!	7
2. Part 1: Cloud Native Development, Microservices and the Architecture of our Covid-19 Data Parser	9
2.1 Agenda	9
2.2 Prerequisites	11
2.3 Microservices	12
3. Part 2: Build your Microservice container with Docker	14
3.1 Agenda	15
3.2 Clone The Repositories	16
3.3 Package Spring Boot with Maven	16
3.4 What is Docker?	16
3.5 Technology vs. Toolkit	17
3.6 Docker Image vs. Docker Container	17
3.7 Building Docker Image from the Dockerfile	19
3.8 Summary	22

4. Part 3: Deploy, Run and Maange your Docker Containers with Kubernetes	23
4.1 Agenda	24
4.2 Kubernetes	24
4.3 Quick reminder about Microservices architecture	25
4.4 Microservices and Kubernetes	25
4.5 Moving from Docker to Kubernetes	26
4.6 Understanding Deployment Scenario in Kubernetes	27
4.7 Kubernetes Concepts/Resources:	27
4.8 Deployment under the hood	27
4.9 Kubernetes Features:	28
4.10 Prerequisites:	29
4.11 What is minikube?	29
4.12 Useful commands through this section:	30
4.13 Useful Commands for Docker	31
4.14 Deploying an Application	31
4.15 Scaling Applications	32
4.16 Exposing an application	33
4.17 Different types of Services for exposing applications	33
4.18 Different types of ports for accessing application from within the cluster, from outside the node and form outside the cluster	34
4.19 Exposing application with type LoadBalancer	34
4.20 Rolling out updates	35
4.21 What is YAML?	37
4.22 Using YAML to create resources	37
4.23 Once YAML file is crafted, here is how to apply it:	40
4.24 Summary	40

5. Part 4: Build, Deploy and Manage your Microservices Application with OpenShift	41
5.1 Agenda	41
5.2 What is OpenShift Container Platform?	42
5.3 3 x key features of OpenShift over Kubernetes. Automation, Agility and Security.	43
5.4 what are the automated workflows?	43
5.5 Three major differences between Kubernetes and OpenShift	44
5.6 Download and Install prerequisites	46
5.7 Login to IBM Cloud and check your installed plugins	47
5.8 Push Image to IBM Container Registry	48
5.9 OC commands	48
5.10 Some useful OC commands	49
5.11 Create Deployment using an image from IBM Cloud Container Registry	49
5.12 Expose the current deployment to the Internet	50
5.13 Pull Images from ICR into non-Default Projects	50
5.14 Verify that the new project can pull images from ICR	51
5.15 Scale and Replicas	51
5.16 Rolling out updates and Rolling back	52
5.17 Summary	54
6. Part 5: Build, Deploy and Share Your Applications with CodeReady Workspaces	55
6.1 Agenda	56
6.2 what is an operator?	59
6.3 What is the OperatorHub:	59
6.4 Install CodeReady Workspaces	61
6.5 Summary	66
7. Part 6: Build, and Test Your Applications with CodeReady Containers	67
7.1 Agenda	67
8. Part 7: Build your CI/CD pipelines with Jenkins and Tekton	70
8.1 Agenda	70

1. Covid-19 Data Analytic Microservices Application with Kubernetes and OpenShift



We have seen a range data published on the impact of various parameters on the spread of covid-19, including population density, average number of people per household, ethnicity, weather data etc. Have you ever wanted to run your own analytics on covid-19 data, and examine data sets in order to draw a particular conclusion? Or possibly evaluate a theory, that may or not may not be true. Such analytics could potentially shed light on the impacts of various factors, and you can apply them to a variety of problems.

Maybe you'd like to see the impact of temperature and humidity on the spread of covid-19 in different countries?

This is a multipart workshop series on building, deploying and managing microservices applications with Kubernetes and openshift.

Our workshop series is around covid-19 data retrieval, parsing and analytic. This is a series of 7 x hands-on workshops, teaching you how to retrieve covid-19 data from an authentic source, make them securely available through REST APIS on kubernetes and Openshift.

The primary applications are developed in Java Spring Boot, but we will add more features and apply analytical services on the data in the form of microservices written in different programming languages.

1.1 Watch the full series on YouTube!



We highly recommend that you follow the workshops by watching the videos as they are hands-on and much more comprehensive than the instructions given here. All videos are available from the links above or directly from this YouTube [playlist](#)

In this workshop series, we will firstly take a look at the key features of our application and how it was developed in microsevices architecture. We'll then explore ways to contianerise our application with Docker. in Lab 3, We'll deploy and manage our application with Kubernetes. In Part 4, we'll deploy our application onto Openshift on IBM Cloud using OpenShift CLI tool and Web Console. In Lab 6, we'll set up a CodeReady Workspace to share an instance of workspace with others with ero configuration on the recipient side. In Lab 7, We'll build and test out application on a local version of Openshift Cluster, CodeReady containers. Finally, in part 8 we'll automate our CI/CD pipeline to push our code into production with zero downtime.

As a reminder, all the steps taught in this course are generic and applicable to application developed in any programming languages or platforms. but to simplify our journey and making it more use-case oriented, our course is designed around a covid-19 data analytic application.

At the beginning of every part, we take a quick look at our application. This is to showcase the end result of what we do together in every part with respect the primary subject of each part.

Our application also comes with a frontend [User Interface](#) that connects to our parsers and invokes the API endpoints to display data and showcase the power of microservices running as containers on Kubernetes and Openshift.

This application has been designed as a template for designing your own analytical microservices and deploying onto Kubernetes.

This workshop series will be focused on:

[Part 1: Cloud Native Development, Microservices and the Architecture of our Covid-19 Data Parser](#)

[Part 2: Build your Microservice container with Docker](#)

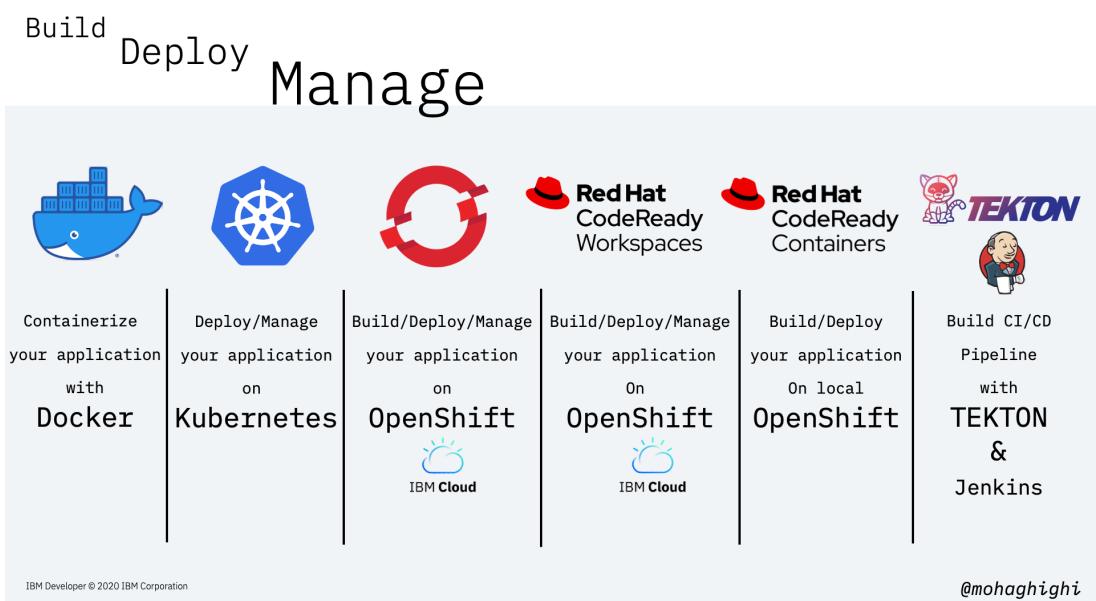
[Part 3: Deploy and manage your application with Kubernetes](#)

[Part 4: Deploy and manage your application with OpenShift on IBM Cloud](#) [Part 5: Build, Deploy and Share with CodeReady Workspaces](#)

[Part 6: Build and Test your application with CodeReady Containers](#)

[Part 7: Build your CI/CD pipelines with Jenkins and Tekton](#)

Here is what you will learn by the end of this workshop series:



2. Part 1: Cloud Native Development, Microservices and the Architecture of our Covid-19 Data Parser

Covid-19 Data Analytics with Kubernetes



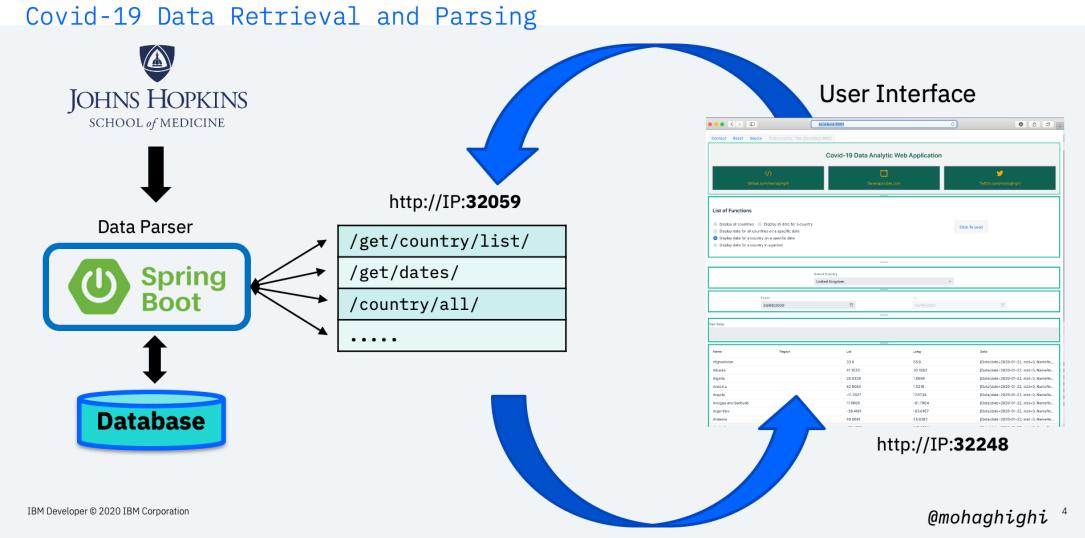
2.1 Agenda

In this section you will learn:

An overview of Covid-19 data analytic web application

- Quick summary
 - Data source & format
 - Data Parser
 - REST APIs endpoints
- Microservices
 - Why microservices?
 - Orchestration with Kubernetes

Overall Architecture



Our application has been developed in Java and Spring Boot framework. It provides us with a number of API endpoints for retrieving covid-19 data per region, country, dates and periods. It comes with a number of containerised microservices, including 2 x data parsers for positive cases and mortality rates per country, and a User Interface for displaying data, as well as invoking those APIs through a number of sample functions.

User Interface

The screenshot shows the 'Covid-19 Data Analytic Web Application' running in a browser. The interface includes a header with social media links (GitHub, Developer.IBM, Twitter) and a 'Connect' button. Below the header is a 'List of Functions' section with radio buttons for different data retrieval options. The 'Display data for a country on a specific date' option is selected. A dropdown menu labeled 'Select Country' shows 'United Kingdom'. Below this are date selection fields ('From: 29/06/2020' and 'To: 29/06/2020') and a 'Raw Data' section. At the bottom is a 'Data display area' containing a table of COVID-19 data for the United Kingdom. Pink arrows point from the text annotations to their corresponding UI elements. The table data is as follows:

Name	Region	Lat	Long	Data
Afghanistan		33.0	65.0	[Data]date=2020-01-22, stat=0, NameRe...
Albania		41.1533	20.1683	[Data]date=2020-01-22, stat=0, NameRe...
Algeria		28.0339	1.6596	[Data]date=2020-01-22, stat=0, NameRe...
Andorra		42.5063	1.5218	[Data]date=2020-01-22, stat=0, NameRe...
Angola		-11.2027	17.8739	[Data]date=2020-01-22, stat=0, NameRe...
Antigua and Barbuda		17.0608	-61.7964	[Data]date=2020-01-22, stat=0, NameRe...
Argentina		-38.4161	-63.6167	[Data]date=2020-01-22, stat=0, NameRe...

[@mohaghghi](http://localhost:8082)

As you can see from the slide, data is fetched from Johns Hopkins University's repo (which is an authentic source of covid-19), and is stored in our local data repository.

Here is a list of sample API endpoints as we'll test them out shortly.

API Endpoints

Method	URI	Return
POST	/get/country/data/{country}	Object
POST	/get/country/data/date/{country},{date}	Object
POST	/get/country/data/dates/{country},{date},{date}	Object
POST	/get/data/date/{date}	Object
GET	/get/country/list/	Object
GET	/get/dates/	Object
GET	/country/all/	Object
GET	/get/source/	String
GET	/hello/	String

@mohaghghi 19

2.2 Prerequisites

Spring Boot v2.2 - <https://spring.io/guides/gs/spring-boot/>

OpenJDK v11 - <https://openjdk.java.net/install/>

(Optional) Apache Netbeans IDE v12 - <https://netbeans.apache.org/download/>

Node.js v14 - <https://nodejs.org/en/download/>

Docker Latest - <https://docs.docker.com/engine/install/>

Minikube Latest - <https://kubernetes.io/docs/tasks/tools/install-minikube/>

CodeReady Containers - <https://developers.redhat.com/products/codeready-containers>

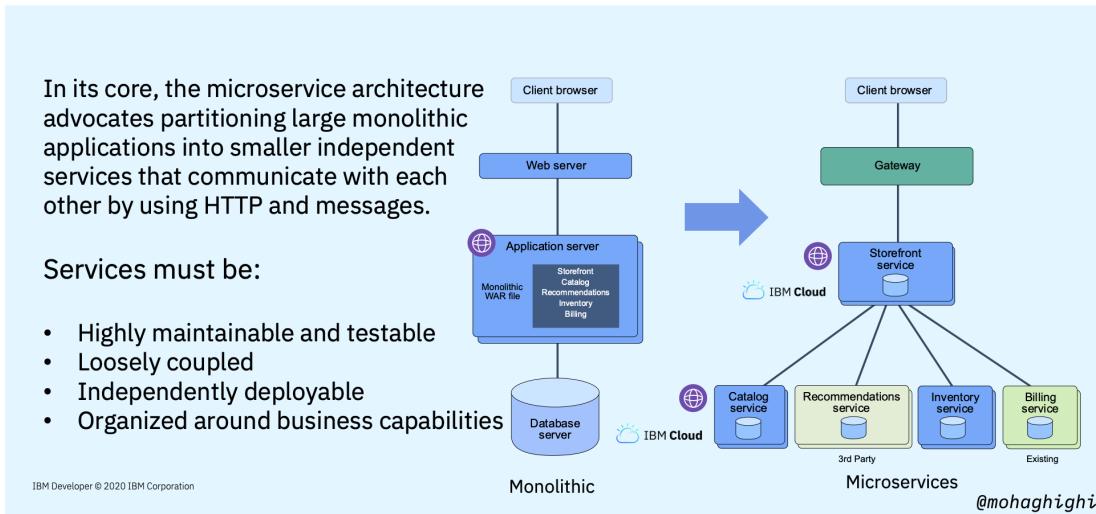
(Optional) OpenShift v4.3 on IBM Cloud - <https://www.ibm.com/cloud/openshift>

Note

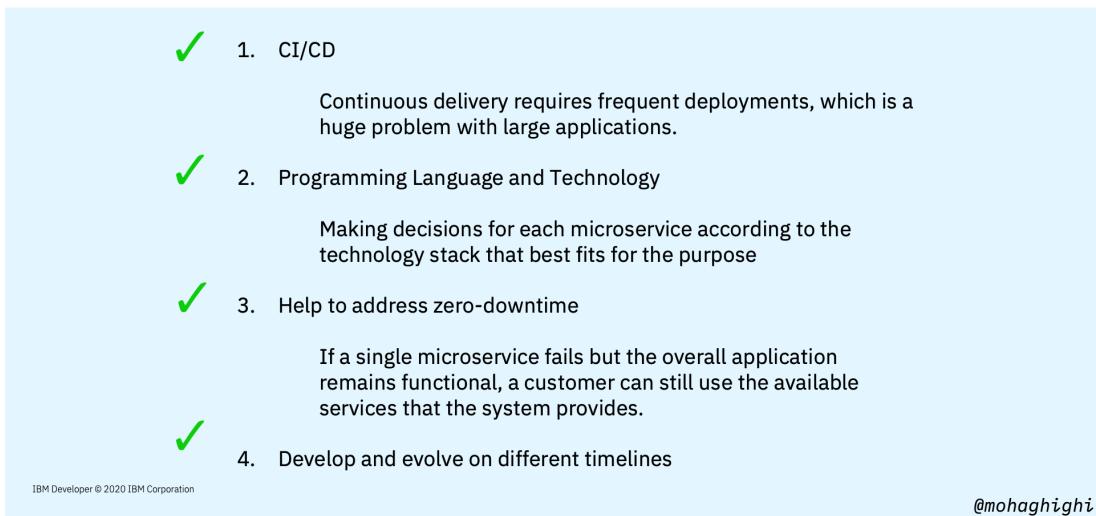
You also need a laptop with a modern operating system (Linux, MacOS or Windows) with at least 16GB memory

2.3 Microservices

Microservices

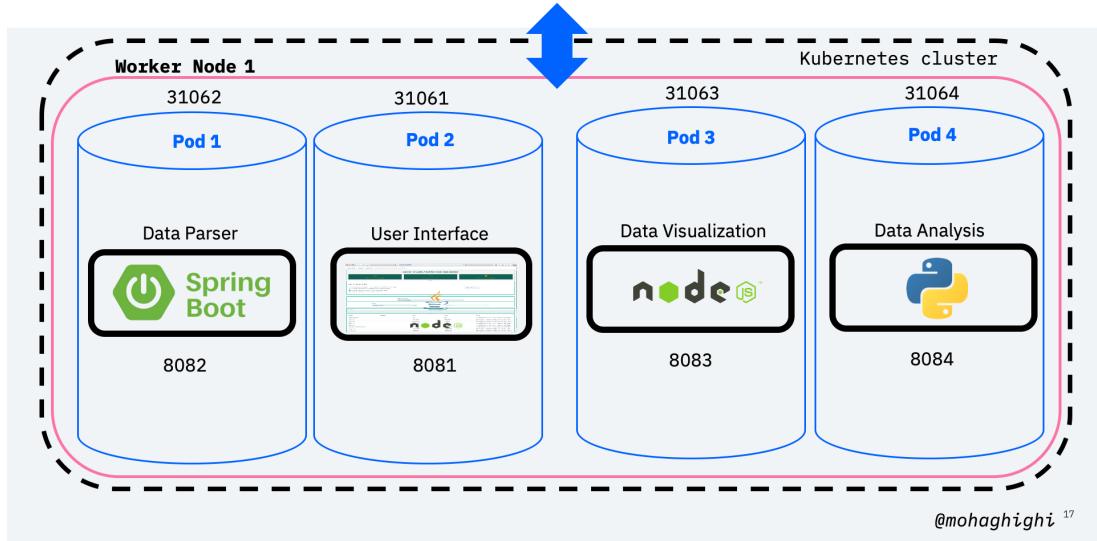


Microservices



By the end of this series, you'll have a microservices application with 4 x containers running in your Kubernetes/OpenShift cluster.

Overall Architecture



- Data Parser written in Java.
- UI frontend written in Java to generate HTML and Node.js.
- Analytical application written in Python Flask.
- Data Visualization application written in Node.js

3. Part 2: Build your Microservice container with Docker

Covid-19 Data Analytics with Kubernetes



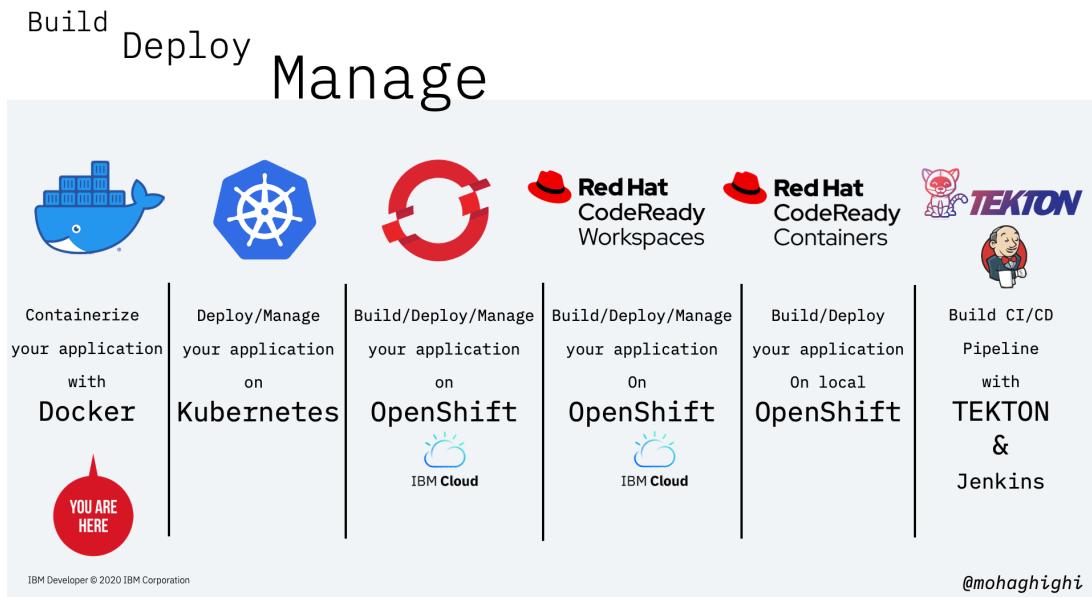
Here's a quick look at what you're going to learn throughout this workshop series - and how Docker fits into our learning journey. In this lab you'll learn about containers, the basics of containerising microservices with Docker, how to run and connect docker containers and best practices for building docker images based on your application services' requirements.

In this lab, we'll containerise our application's microservices with Docker, and in the next lab, we'll deploy and manage them with Kubernetes. Later we'll use openshift to automate the entire process of containerising, deployment, scaling and management with a few clicks from the openshift web console.

3.1 Agenda

In this section you will learn:

- Install/download prerequisites
- Package Java Maven application
- Test Java application
- Docker
 - Dockerfile
 - Build Docker image
 - Run Docker containers
 - Use Kubernetes Docker daemon
 - Docker Registry
 - SSH into Docker images
 - Connecting Docker containers
 - Inspect Docker Containers



In the previous labs, we broke down our application into several microservices based on their functionalities and purposes, and in this lab we'll containerise them with Docker, and use docker to run them.

Therefore, we convert our monolithic application into a multi-container application.

If you want to review how this application has been designed and how microservices architecture optimised it, please refer to the previous workshop.

You may ask why Docker?

Well, Modern application development and app modernisation techniques consist of three important stages of Build, Deploy and Manage.

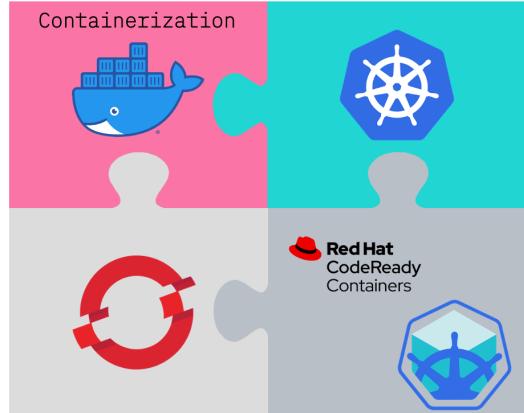
Docker plays a vital role in the build stage, and even partially the deployment phase.

As you can see from this slide, for stages we're going to follow in this workshop series, Docker is responsible for all initial steps.

Build/Deploy/Manage

Containerize with Docker

- I. Package Spring App
- II. Craft Dockerfile
- III. Build Image
- IV. Push to Registry
- V. Pull from Registry
- VI. Deploy onto Kubernetes
- VII. Create Deployments
- VIII. Create Pods
- IX. Adding Services
- X. Scaling
- XI. Rolling updates
- XII. Manage Resources



IBM Developer © 2020 IBM Corporation

@mohaghichi

Let's start by cloning the repos and packaging our Java application with Maven:

3.2 Clone The Repositories

```
git clone https://github.com/mohaghichi/covid19-web-application
git clone https://github.com/mohaghichi/covid19-UI
```

3.3 Package Spring Boot with Maven

```
./mvnw clean install
```

Run the jar file to test the Spring Boot application:

```
java -jar target/[filename].jar
```

Data Parser runs on port 8082. if you want to change the *Port Number*, you need to edit "*application.properties*" file under src/main/java/resources/

```
curl http://localhost:8082
```

Now we've got our application ready to be containerised with Docker. Before we dive deeper into Docker, let's explore what containers are and how docker fits in containerisation technology.

3.3.1 What is a container?

Containers are executable units of software in which application code is packaged, along with its libraries and dependencies, in common ways so that they can be run anywhere, whether it be on desktop, traditional IT, or the cloud.

3.4 What is Docker?

“Docker is the de facto standard to build and share containerized apps - from desktop, to the cloud”

You may ask why Docker?

Modern application development and app modernisation techniques consist of three important stages of Build, Deploy and Manage.

Docker plays a vital role in the build stage, and even partially the deployment phase.

As you can see from this slide, for stages we're going to follow in this workshop series, Docker is responsible for all initial steps.

3.5 Technology vs. Toolkit

Containers have been around for quite some time, and developers can create containers without Docker – but Docker makes it easier, simpler, and safer to build, deploy, and manage containers. Docker is essentially the first toolkit that due to its simplicity, enabled all developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation.

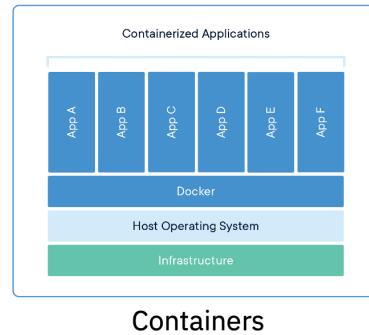


What is Docker?

“Docker is the de facto standard to build and share containerized apps - from desktop, to the cloud”

“Technology” vs. “Toolkit”

Docker is an open source containerization platform for packaging applications into containers.
It's essentially a toolkit that enables developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation.



@mohaghghi

3.6 Docker Image vs. Docker Container

Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. (only interacting with designated resources)

Container *images become containers at runtime* and in the case of Docker containers - images become containers when they run on Docker.

So let's get started and build our first container image with Docker.

The first step is to craft our dockerfile and the Dockerfile is essentially the build instructions to build the image.



What is Docker?

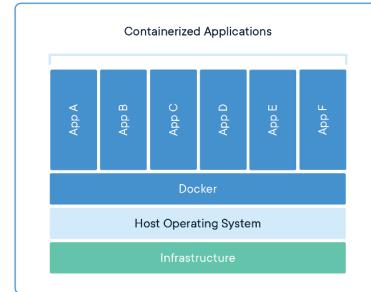
Docker is the de facto standard to build and share containerized apps - from desktop, to the cloud"

"Binary file" vs. "Processes"

Container is a standard unit of software that packages up code and all its dependencies.

Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. (only interacting with designated resources)

Container images become **containers** at runtime and in the case of **Docker containers** - images become containers when they run on Docker.



Containers

@mohaghichi

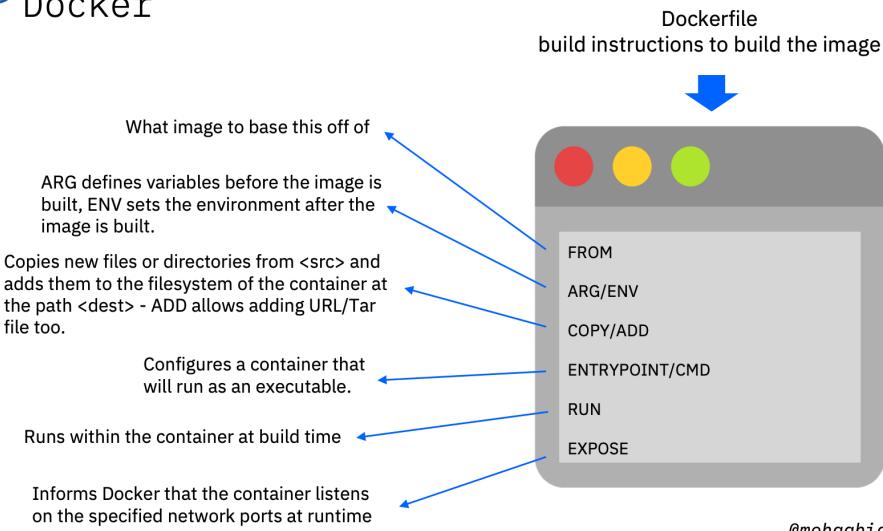
3.6.1 What is a Dockerfile?

A set of build instructions to build the image in a file called "dockerfile".

3.6.2 Craft your Dockerfile



Docker



The first part is the FROM command, which tells docker what image to base this off of. The FROM instruction sets the Base Image for subsequent instructions. It'll start by pulling an image from the Public Repositories.

ARG defines instructions to define variables. ENV is similar to ENV but mainly meant to provide default values for your future environment variables. ARG values are not available after the image is built.

The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>. It can copy a file (in the same directory as the Dockerfile) to the container

The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

The ENV instruction sets the environment variable <key> to the value <value>.

This is what runs within the container at build time. The RUN instruction will execute any commands in a new layer on top of the current image and commit the results.

An ENTRYPOINT allows you to configure a container that will run as an executable.

Note

should add '&' to run in the background

[Entry Point/CMD] ENTRYPOINT instruction allows you to configure a container that will run as an executable. It looks similar to CMD, because it also allows you to specify a command with parameters. The difference is ENTRYPOINT command and parameters are not ignored when Docker container runs with command line parameters.

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. The EXPOSE instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.

In the case of our Data Parser Spring Boot application:

```
FROM adoptopenjdk/openjdk11:latest
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Dockerfile for Node.js application:

```
FROM node:12
COPY package.json .
RUN npm install
ENTRYPOINT ["node", "app.js"]
```

Dockerfile for Python application:

```
FROM python:3
COPY package.py .
RUN pip install pystrich
ENTRYPOINT ["python", "./app.py"]
```

save the file as dockerfile with no file extension.

3.7 Building Docker Image from the Dockerfile

```
docker build -t [image name:v1] [path]
```

in this case, let's call it myapp:v1

```
docker build -t myapp:v1 .
```

let's take a look at our docker images:

```
docker images
```

our image must be listed there.

now let's a look at running containers:

```
docker ps
```

if you add -al, you can view all running and stopped containers

```
docker ps -al
```

Here's the command for running the docker container

```
docker run -p [PortHost:PortContainer] [imageName] -d --rm
```

Now let's go ahead and run our container on port 8082:

```
docker run -p 8082:8082 myapp:v1 -d
```

-d and --rm flags will respectively run the docker in detached mode and replace an existing docker image of the same name with the name one. We can ping the application by invoking the /hello/ REST endpoint:

```
curl localhost:8082/hello/
```

3.7.1 Build and Run the UI App

The UI application can be retrieved from here: <https://github.com/mohaghghi/Covid19-UI.git>

Now let's build the UI app and call it myui:v1 Dockerfile is the same as the one we used for Data Parser app but changing the name to "*myui*"

```
docker build -t myui:v1 .
```

in case you haven't run the maven build and packaged the UI App, run this where mvnm file is located

```
./mvnm clean install
```

Now let's run the UI app on port 8081:

```
docker run -p 8082:8082 myapp:v1 -d
```

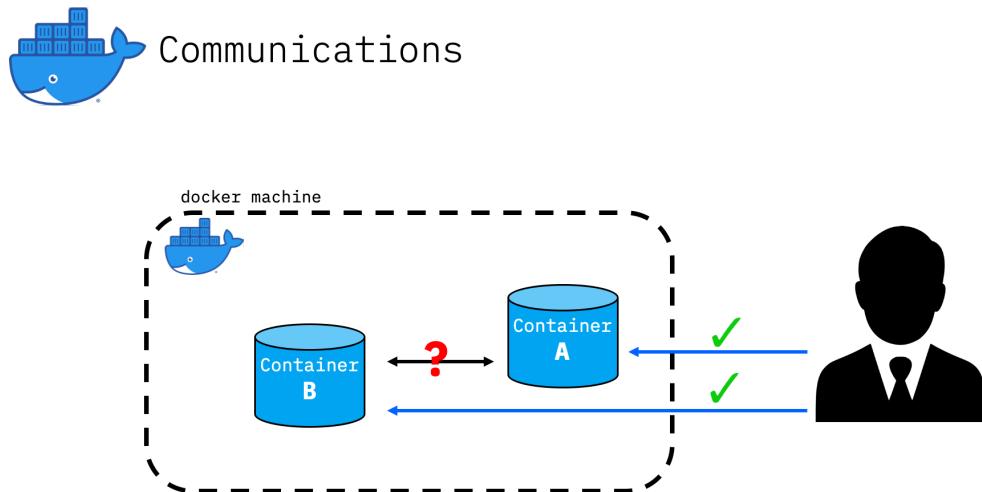
Open your browser and navigate to

```
localhost:8081
```

From the UI, click on connect on the top left hand corner and enter:

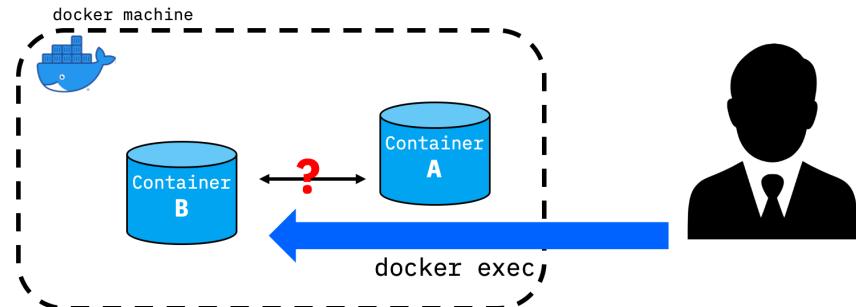
```
http://localhost:8082
```

As you may have seen, you got an error indicating that the server is not responding. There reason is, we can connect to containers directly through Docker, but docker containers cannot discover or communicate with each other.



45
@mohaghghi

now let's try to ssh into our one of the docker containers and try to connect to the other one to identify the problem. To simulate the issue that we've just experienced with the UI app, let's ssh into our UI and try to connect to our data parser from within that container.

46
@mohaghghi

```
docker exec [container name/ID] -it
```

Here how we ssh into UI app

```
docker exec -it myui:v1 /bin/bash
```

Now let's connect from within the container and see if it works

```
curl localhost:8082/hello/
```

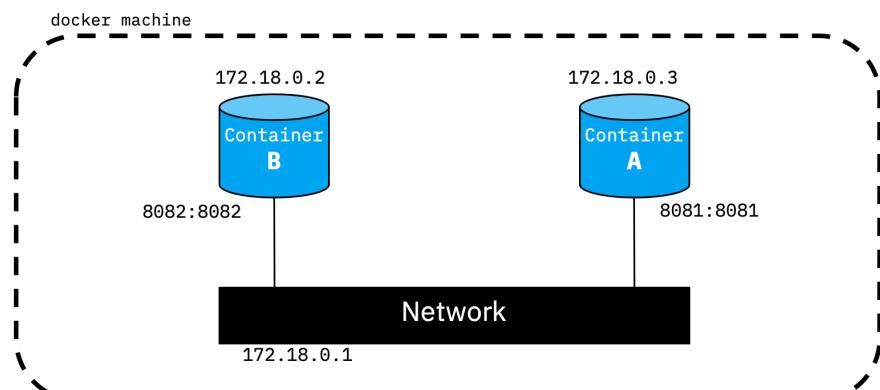
As you can see that doesn't work either.

Containers need to be connected to the same network in order to communicate with each other

You can inspect your container to investigate the matter by looking for the network within both containers.

```
docker inspect [container name]
```

As you can see our UI and Parser apps are not part of the same network.

48
@mohaghghi

Let's create a network and instruct our containers to connect to it

```
docker network create test
```

let's stop our docker containers:

```
docker stop [container id]
```

Let's run our containers again, this time instructing them to join the new network we've just created

```
docker run -p [PortHost:PortContainer] [imageName] --net=test
```

Run UI application on test network:

```
docker run -p 8081:8081 myui:v1 --net=test
```

Run parser application on test network:

```
docker run -p 8082:8082 myapp:v1 --net=test
```

Let's inspect our containers again and get their IP addresses based on their new network

```
docker inspect [container name/ID]
```

if we try to ping our applications again, they should work fine.

Go ahead and connect to the parser form the UI app to verify that.

In the next part we will be using minikube to spin up a single node kubernetes cluster. If we build all our images on your host docker machine, it'd be quite difficult to transfer your images from your host into minikube.

one solution is to use minikube's docker daemon to build your docker images.

you need to set your environmental parameter to use miinkube docker. This command will let you do that:

```
eval $(minikube docker-env)
```

This step is not needed here, is intended to let you know what we will use minikube's docker.

3.8 Summary

Summary

Part Two

- Containers
- Docker
 - Dockerfile
 - Build Docker image
 - Run Docker containers
 - Use Kubernetes Docker daemon
 - Docker Registry
 - SSH into Docker images
 - Connecting Docker containers
 - Inspect Docker Containers

IBM Developer © 2020 IBM Corporation

@mohaghghi

4. Part 3: Deploy, Run and Maange your Docker Containers with Kubernetes

Covid-19 Data Analytics with Kubernetes



4.1 Agenda

In this section you will learn:

- Why Kubernetes

- Kubernetes concepts/components

- Deploy on Kubernetes

- Minikube

- Pulling image from registry

- Create deployment

- Expose deployment

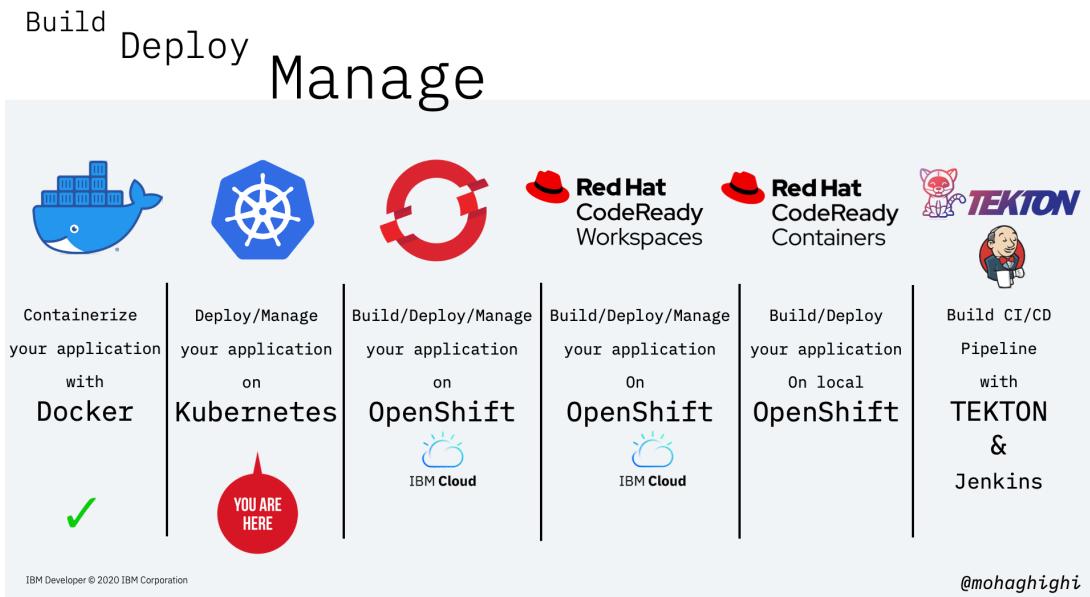
- Create services

- Manage with Kubernetes

- Replicasets

- Rolling out updates

- Autoscaling



4.2 Kubernetes

Kubernetes is Greek for helmsman or pilot, hence the helm in the Kubernetes logo.

Imagine a ship full of containers like in this photo, and the helmsman is to make sure the ship sails smoothly through the oceans, and despite all the tides and waves, it makes it to the destination safely. The helmsman orders his crew to evenly distribute the containers around the ship in a way that, proper balance is struck, no one side is abnormally heavier, containers won't fall off, and the ship sails smoothly throughout the journey.

Just like the helmsman, Kubernetes looks after a large number of containerised applications, by orchestrating them according to the load, and the available underlying resources, making sure our system achieves minimum zero downtime and our applications are always up and running.

In the first and second labs we learned about the advantages and motivations for moving away from Monolithic applications and adopting microservices architecture.

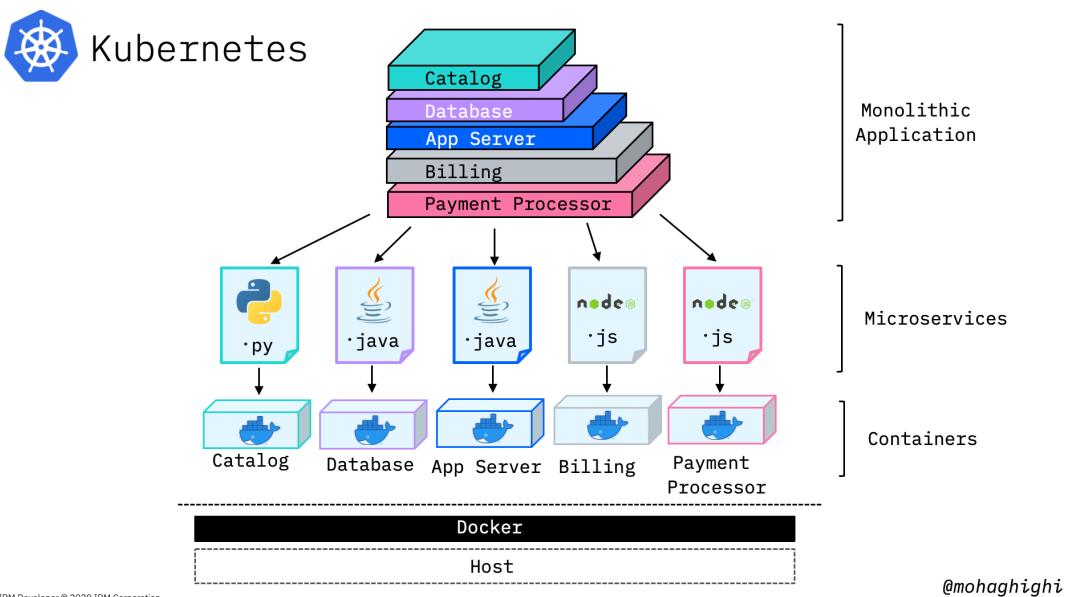
4.3 Quick reminder about Microservices architecture

Microservices architecture addresses all of the liabilities that are inherent in monolithic applications. microservices architecture allows

1. Different parts of our application to evolve on different timelines,
2. They can be deployed separately,
3. You choose your technology stack for each Microservice as it best fits the purpose,
4. You can scale your services dynamically at runtime. Or let's say you can create individual instances of each individual service.

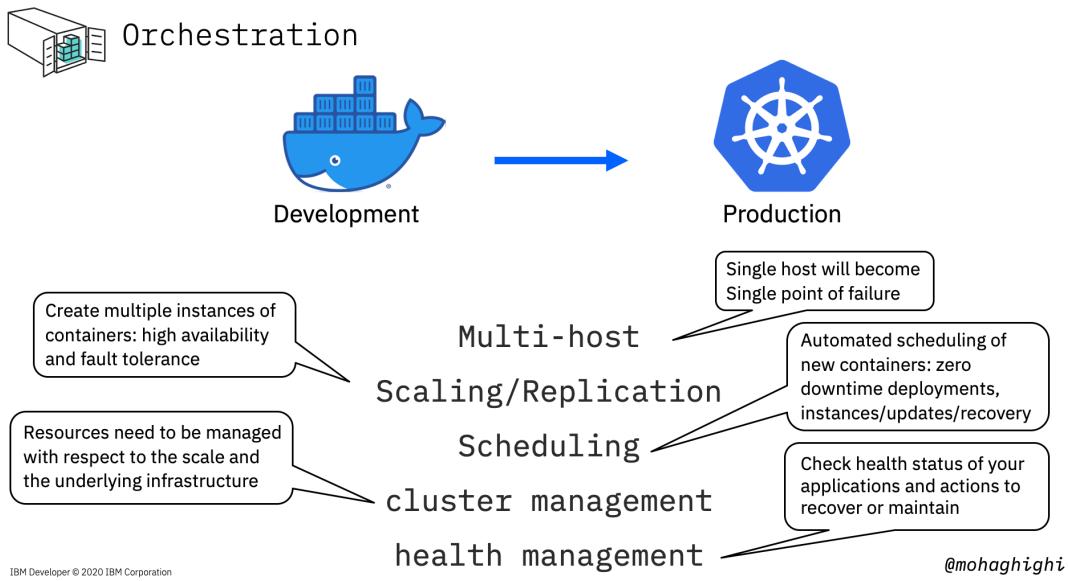
But the most obvious advantage here is, if any part of the application fails, the whole application will not necessarily become unavailable/unresponsive to the customer, because they are not designed and operated as a single entity like in monolithic architecture.

4.4 Microservices and Kubernetes



In the previous labs, we broke down our application into several microservices and then containerised them with Docker and let docker run them. So we converted our application into a multi-container application in order to remove that single point of failure. But here 's the problem: Docker is running on a single host.

4.5 Moving from Docker to Kubernetes

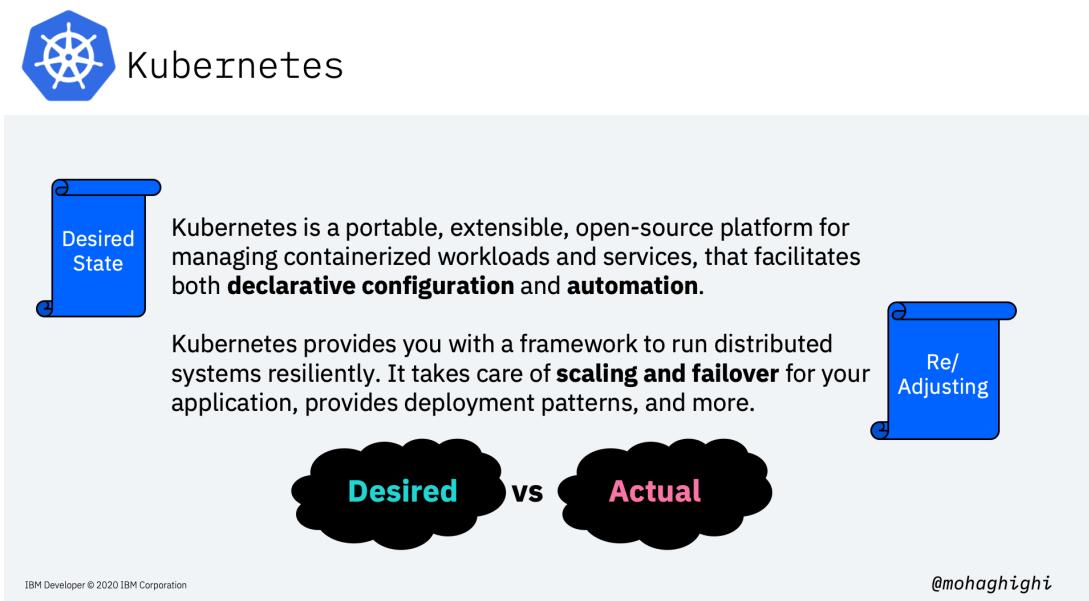


And here we discuss why we need a containers orchestration platform like Kubernetes when moving from development to production.

a multi-container application must run on a multi-host environment in order to eliminate that single point of failure. If one host went down our orchestration tool can switch the load to another host.

We need to be able to create new instances of our individual microservices containers to scale accordingly.

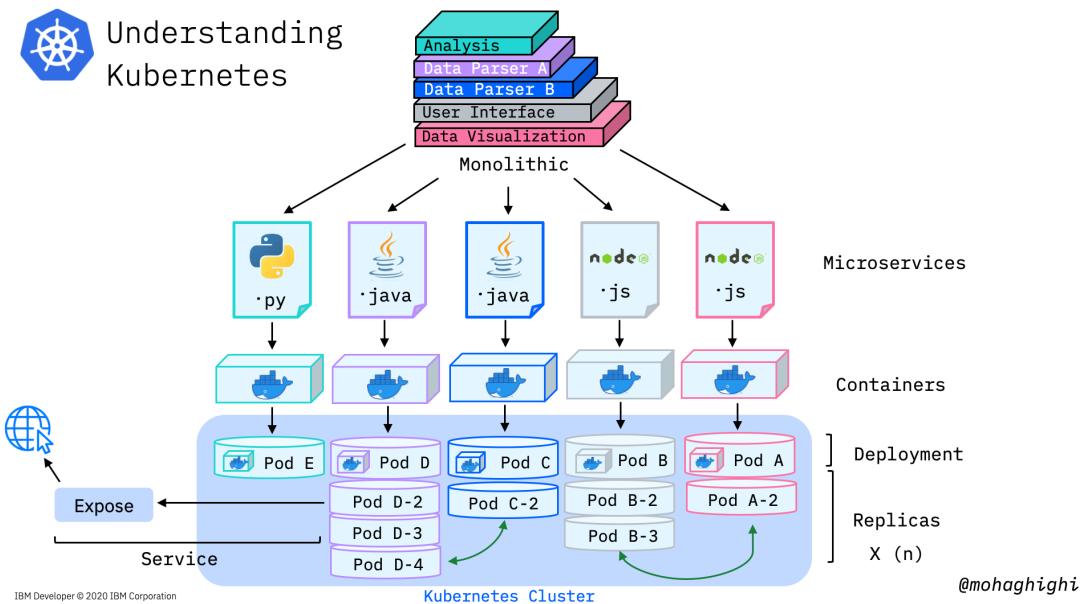
When one or more of our services need to be updated, or let's say we are adding a new service to our mix, the orchestration platform must be able to automatically schedule new deployments and create new instances of our containers with zero downtime.



Kubernetes scales and manages our containers according to the available underlying resources on the host. Docker has a good view of what's happening to our containers, but not our host machine.

Last but not least, Kubernetes checks our container continually to make sure they're healthy, and in case of any failure, it'll take actions to reinstate our deployment, create new instances or restore the services.

4.6 Understanding Deployment Scenario in Kubernetes



Now let's take a look at a deployment scenario on a high level, how we are going to deploy our application onto Kubernetes.

We broke down our application, built docker containers, deploying each docker container will spin up a pod with its docker container in there. Based on our deployment scenario, and the load, each pod gets replicated (and that way we're making new instances of the docker containers) -these pods are inside a worker, which we are showing them for simplicity. so we first created a deployment, and then scale our deployment accordingly. Next step is to create a service, which allows our applications communicate with each within the cluster and also exposes our application to the internet and external networks. If the service type is a load balancer, Traffic coming to our application will be directed to the pods accordingly through the load-balancer service.

4.7 Kubernetes Concepts/Resources:

Pod: Group of one or more containers with shared storage/network and a specification for how to run the containers in a shared context.

Deployment: A set of multiple, identical Pods with no unique identities. It runs multiple replicas of your application, and automatically replaces any failed instances.

Node: A virtual or a physical machine with multiple pods, where Master node automatically handles scheduling the pods across the Worker nodes in the cluster.

Service: An abstraction which defines a logical set of Pods and a policy by which to access them. Service enables external access to a set of Pods.

Label: Labels are key/value pairs that are attached to objects, such as pods.

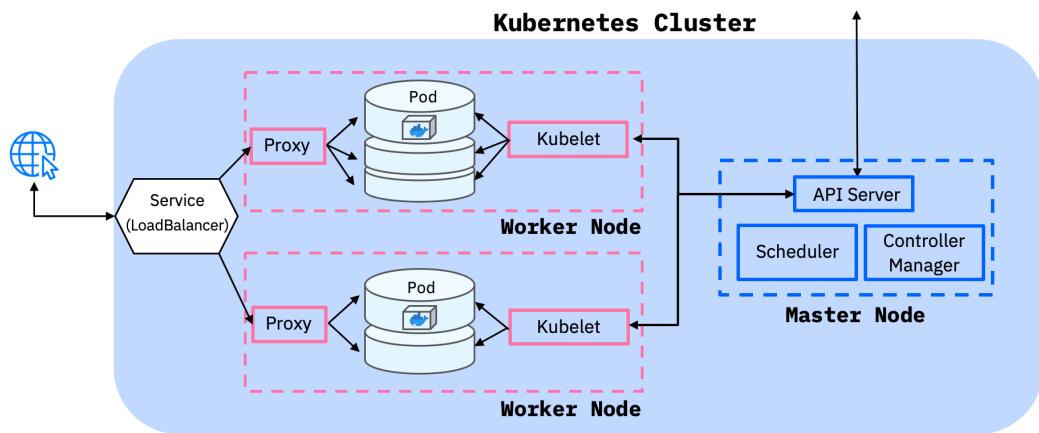
Namespace: Logical isolation/partitioning of resources in kubernetes cluster.

Now that we know the key components, let's revisit our deployment scenario, this time in more details to see what's happening under the hood.

4.8 Deployment under the hood

Firstly, we'll use KUBECTL CLI tool to interact with Kubernetes cluster. The kubectl lets you control Kubernetes clusters and its resources.

Think of kubectl as your magic keyword to instruct Kubernetes from your terminal.



IBM Developer © 2020 IBM Corporation

@mohaghghi

4.9 Kubernetes Features:

- Automated rollouts and rollbacks
- Automatic scaling and load balancing
- Self-healing
- Service discovery
- Storage orchestration

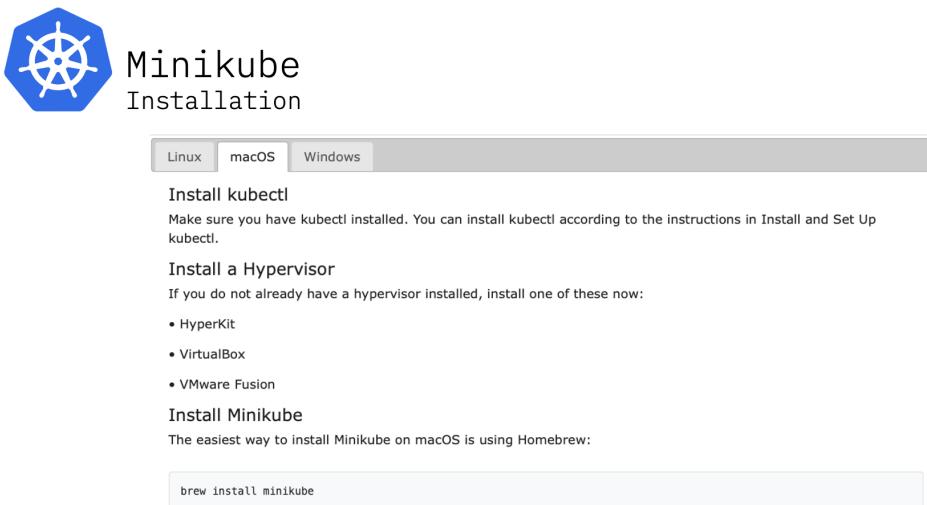
Automated rolling out changes to a deployment and the ability to pause, resume and rollback to previous version if needed.

Automatic scaling and load balancing: When traffic to a container spikes, Kubernetes can employ load balancing and scaling to distribute it across the network to maintain stability.

Self-healing: When a container fails, Kubernetes can restart or replace it automatically; it can also take down containers that don't meet your health-check requirements.

Service discovery: Kubernetes can automatically expose a container to the internet or to other containers using a DNS name and IP address.
And finally, provisioning local or cloud storage for your containers as needed.

4.10 Prerequisites:



The screenshot shows the 'Minikube Installation' guide. It features a blue hexagonal icon with a white steering wheel in the top left. The title 'Minikube Installation' is centered above a navigation bar with tabs for 'Linux', 'macOS' (which is selected), and 'Windows'. Below the tabs, there are three sections: 'Install kubectl', 'Install a Hypervisor', and 'Install Minikube'. Under 'Install kubectl', it says to make sure kubectl is installed and provides links for 'Install' and 'Set Up kubectl'. Under 'Install a Hypervisor', it lists 'HyperKit', 'VirtualBox', and 'VMware Fusion' as options. Under 'Install Minikube', it says the easiest way on macOS is via Homebrew with the command 'brew install minikube'.

IBM Developer © 2020 IBM Corporation

@mohaghghi

In this part we are going to use minikube to spin up a single-node kubernetes cluster locally.

Here's the link to minikube on your machine:

```
https://kubernetes.io/docs/tasks/tools/install-minikube/
```

4.11 What is minikube?



The screenshot shows the 'Minikube' landing page. It features a large blue hexagonal icon with a white steering wheel in the center. The word 'Minikube' is written in a bold, sans-serif font to the right of the icon. Below the icon, there is a short paragraph of text describing what Minikube is: 'Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day.'

Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

IBM Developer © 2020 IBM Corporation

@mohaghghi

4.11.1 Spin up a Kubernetes cluster

```
minikube start
```

4.11.2 Start minikube by limiting the resources' utilization

```
minikube start --memory=8192 --cpus=3 --kubernetes-version=v1.17.4 --vm-driver=virtualbox
```

4.11.3 Get cluster information

```
kubectl cluster-info
```

4.11.4 Get cluster configuration

```
kubectl config view
```

4.12 Useful commands through this section:

4.12.1 Get the list of Pods

```
kubectl get pods
```

4.12.2 Get the list of Deployments

```
kubectl get deployment
```

4.12.3 Pause minikube

```
kubectl pause minikube
```

4.12.4 Stop minikube

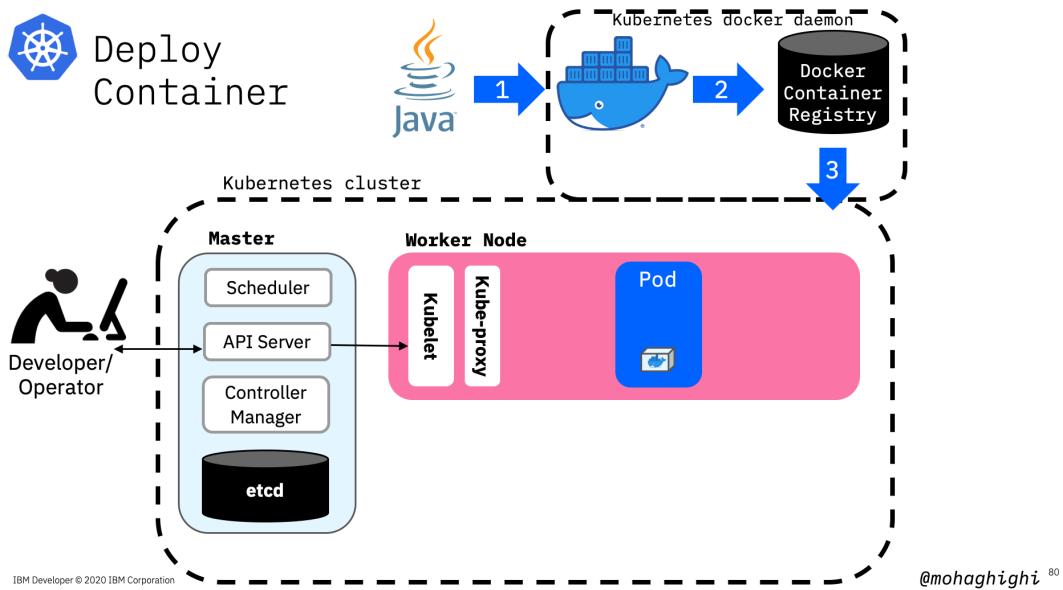
```
kubectl stop minikube
```

4.12.5 Starting Kubernetes dashboard

```
kubectl minikube dashboard
```

4.12.6 set minikube docker daemon

```
eval $(minikube docker-env)
```



4.12.7 Verify you're using minikube's docker by looking up the images

```
docker get images
```

4.13 Useful Commands for Docker

4.13.1 Getting the list of containers

```
docker container list
```

4.13.2 Getting running docker containers

```
docker ps
```

4.14 Deploying an Application

4.14.1 Creating deployment with an image

```
kubectl create deployment [label] --image= [Image Name]
```

4.14.2 Getting details on deployment

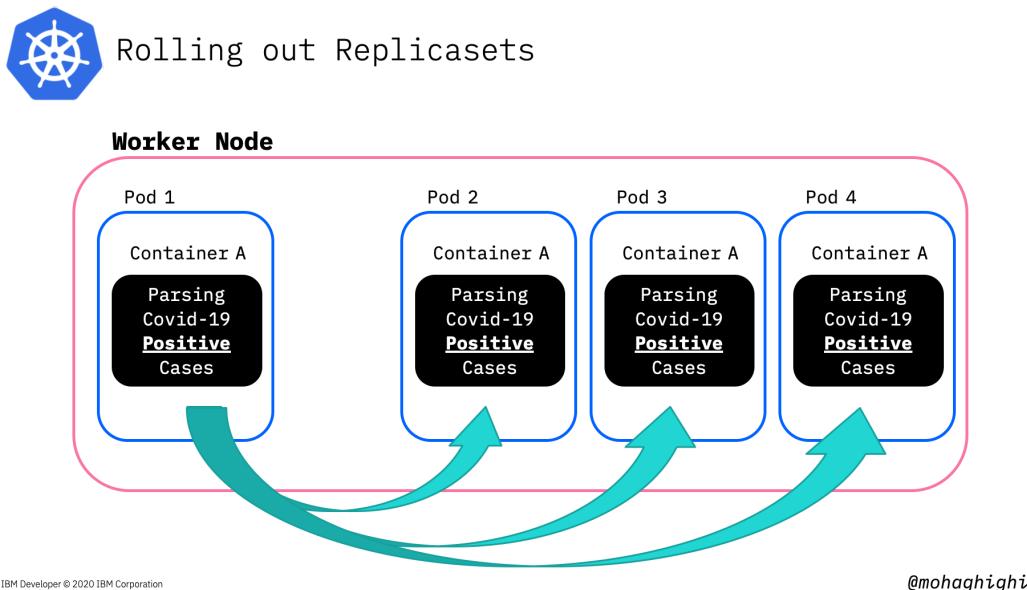
```
kubectl describe deployment/[deployment]
```

4.14.3 Getting logs for deployment

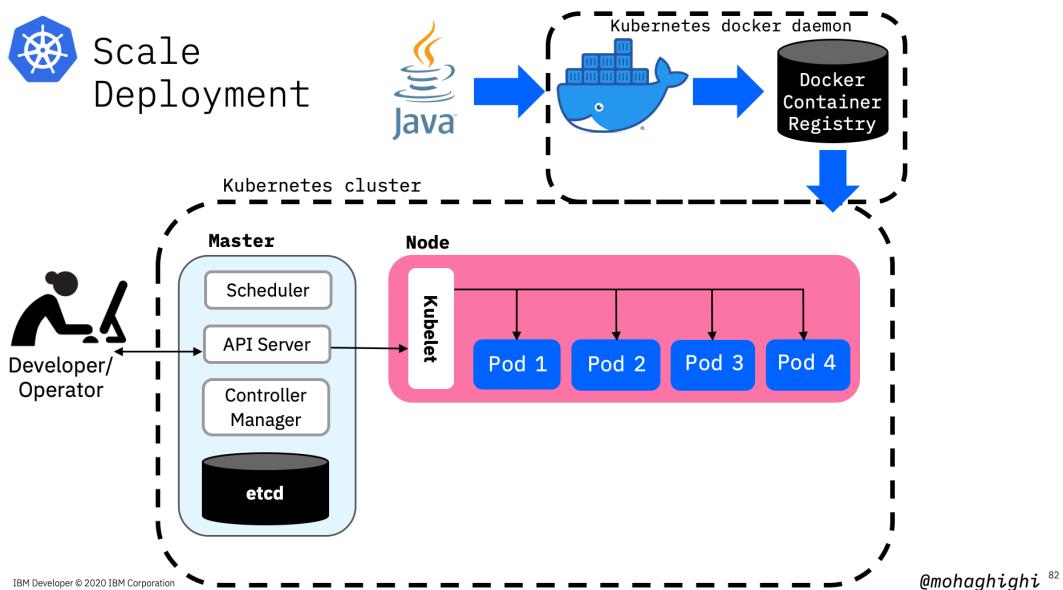
```
kubectl get events
```

4.15 Scaling Applications

4.15.1 creating instances of the application by setting the replicas



4.15.2 Creating replicas and the processes under the hood



4.15.3 Scale deployment and setting replicas

```
kubectl scale deployment [Deployment Name] --replicas=4
```

4.15.4 Enabling application to automatically scale

```
kubectl autoscale deployment [deployment] --min=1 --max=8 --cpu-percent=80
```

4.15.5 Getting Info on Horizontal Pod Autoscaler

```
kubectl get hpa
```

4.16 Exposing an application

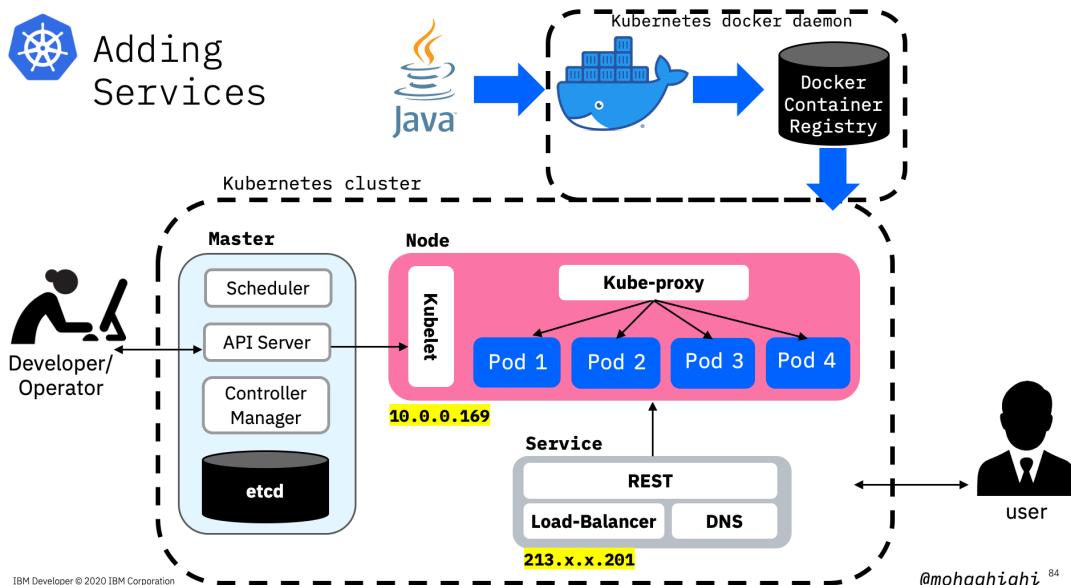
```
kubectl expose deployment [deployment Name] [--port=8082] --type=NodePort
```

4.16.1 Getting list of services

```
kubectl get services
```

4.16.2 Pinging the application

```
curl [Master IP]:[NodePort]/hello/
```



4.16.3 ssh into kubernetes cluster to ping the pod from within the cluster

```
minikube ssh
```

4.16.4 Ping the container

```
curl [Pod IP]:[container port]/hello/
```

4.17 Different types of Services for exposing applications

ClusterIP: This default type exposes the service on a cluster-internal IP. You can reach the service only from within the cluster.

NodePort: This type of service exposes the service on each node's IP at a static port. A ClusterIP service is created automatically, and the NodePort service will route to it. From outside the cluster, you can contact the NodePort service by using "<NodeIP>:<NodePort>".

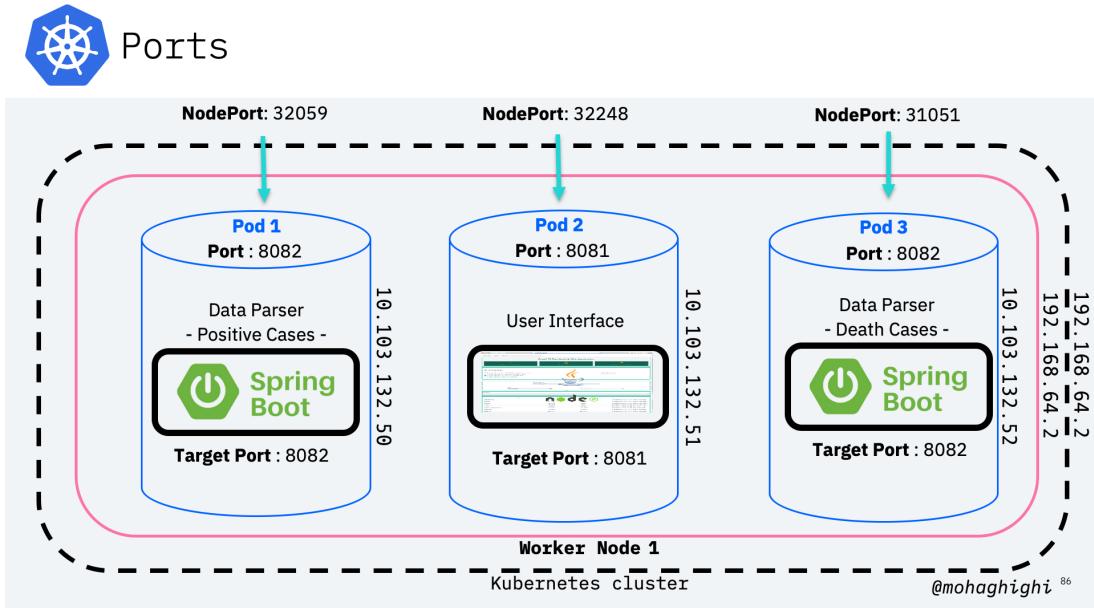
LoadBalancer: This service type exposes the service externally using the load balancer of your cloud provider. The external load balancer routes to your NodePort and ClusterIP services, which are created automatically.

4.18 Different types of ports for accessing application from within the cluster, from outside the node and from outside the cluster

NodePort: This setting makes the service visible outside the Kubernetes cluster by the node's IP address and the port number declared in this property. The service also has to be of type NodePort (if this field isn't specified, Kubernetes will allocate a node port automatically).

Port: Expose the service on the specified port internally within the cluster. That is, the service becomes visible on this port, and will send requests made to this port to the pods selected by the service.

TargetPort: This is the port on the pod that the request gets sent to. Your application needs to be listening for network requests on this port for the service to work.



4.19 Exposing application with type LoadBalancer

```
kubectl expose deployment [deployment Name] [--port=8082] --type=LoadBalancer
```

4.19.1 Getting the Cluster-IP for the Kubernetes Cluster

```
kubectl cluster-info
```

4.19.2 This command doesn't work as Minikube doesn't allocate the external IP address

```
curl [LoadBalancer External IP]:[Node Port]/hello/
```

Info

minikube is a single node cluster. therefore its IP address is the same node IP

4.19.3 Pinging the container using minikube cluster IP instead worker node IP and NodePort

```
curl [kubernetes Cluster-IP]:[Node Port]/hello/
```

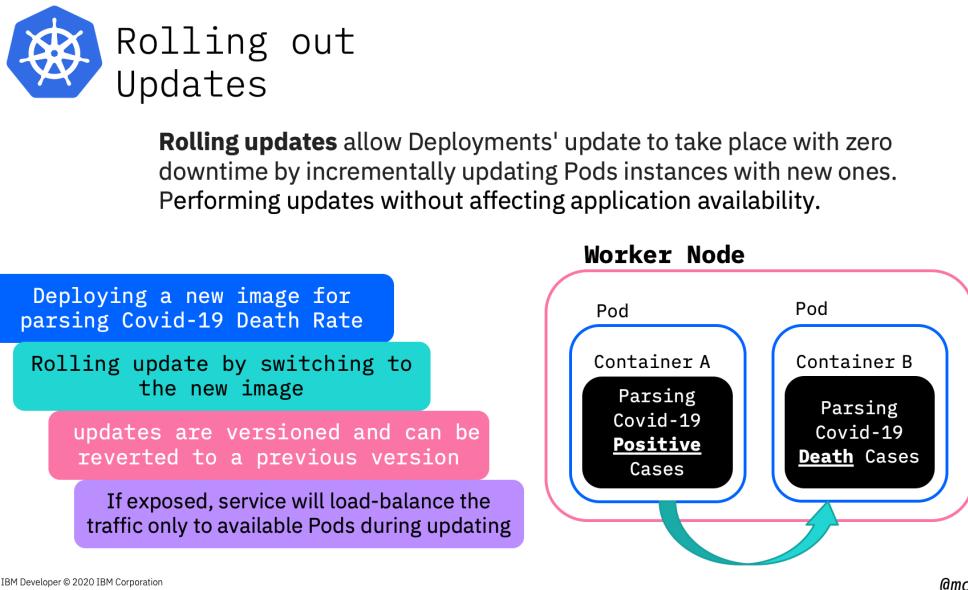
4.19.4 Now let's try to access the pod from within the cluster

```
minikube ssh
```

4.19.5 Using the Load Balancer IP and container Port

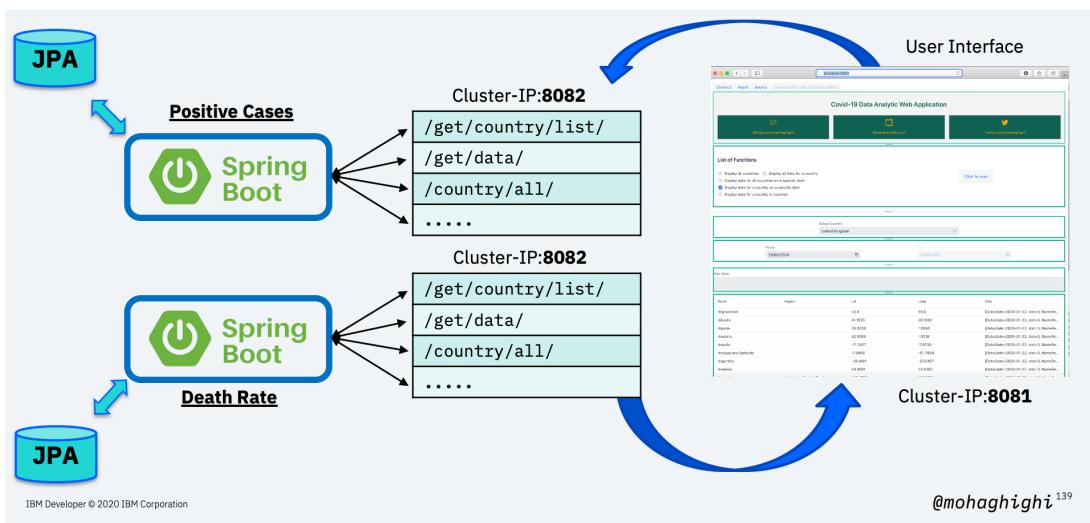
```
curl [LoadBalancer Cluster IP(internal)]:[Port]/hello/
```

4.20 Rolling out updates



Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. Performing updates without affecting application availability.

Overall Architecture



In this part we're going to update our image to the parser for covid-19 mortality data, which reflects the number of death in every country country and region.

```
kubectl set image deployment/[deployment name] [container]=[new image]
```

Make sure you use the container name in the above command to update the image in it.

To get the container name, use:

```
kubectl get deployment -o wide
```

verify the deployment is updated by pinging the app

```
curl ip:port/hello/  
curl ip:port/get/country/data/germany/
```

To rollback to the previous version use:

```
kubectl rollout undo deployment/[deployment Name]
```

optional: You can add --to-revision=n in order to rollback to a specific version

```
kubectl rollout undo deployment/[deployment Name] --to-revision=2
```

checkout the rollout status

```
kubectl rollout status deployment/[deployment Name]
```

4.21 What is YAML?

YAML is a human-readable, data serialization standard for specifying configuration-type information. YAML can be used for common use cases such as:

- Configuration files
- Inter-process messaging
- Cross-language data sharing



Structure of YAML

Key Value Pair – The basic type of entry in a YAML file is of a key value pair. After the Key and colon there is a space and then the value.

Arrays/Lists – Lists would have a number of items listed under the name of the list. The elements of the list would start with a '-' .

Dictionary/Map – A more complex type of YAML file would be a Dictionary and Map

Key Value Pair	Array/Lists	Dictionary/Map
Fruit: Apple Vegetable: Radish Liquid: Water Meat: Goat	Fruits: - Orange - Banana - Mango Vegetables: - Potato - Tomato - Carrot	Banana: Calories: 200 Fat: 0.5g Carbs: 30g Grapes: Calories: 100 Fat: 0.4g Carbs: 20g

IBM Developer © 2020 IBM Corporation

@mohaghighi ⁹⁵

Kubernetes resources are represented as objects and can be expressed in **YAML** or **JSON** format Examples:

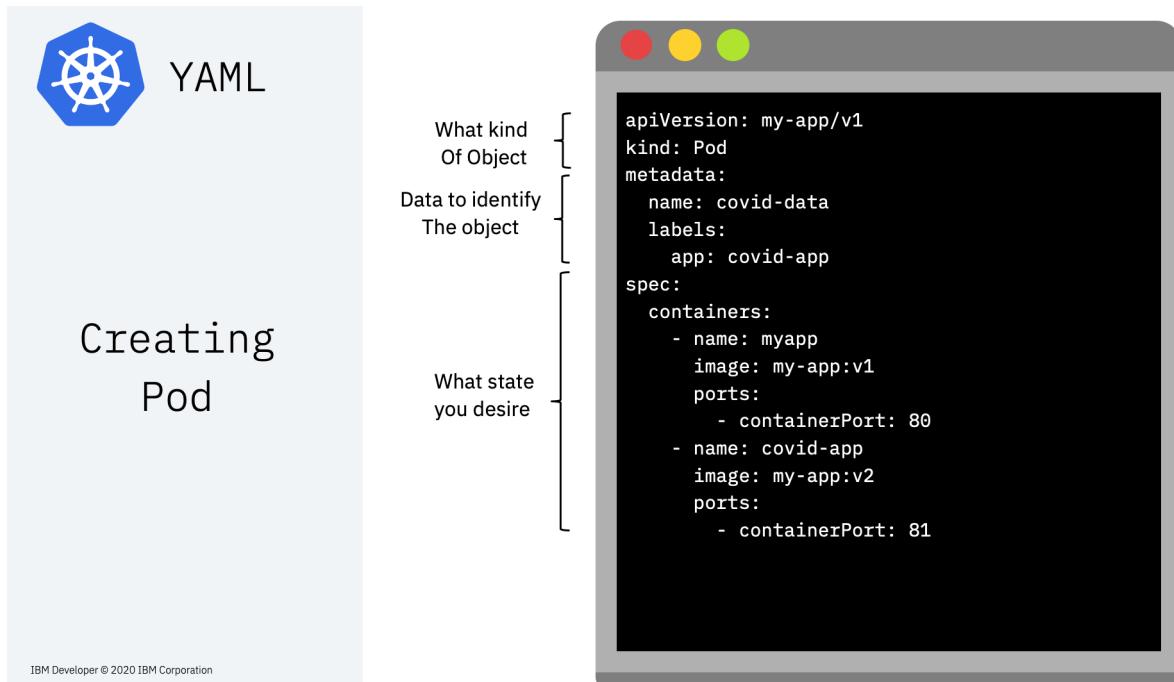
- Print deployment as Yaml

```
kubectl get deployment -o yaml [json]
```

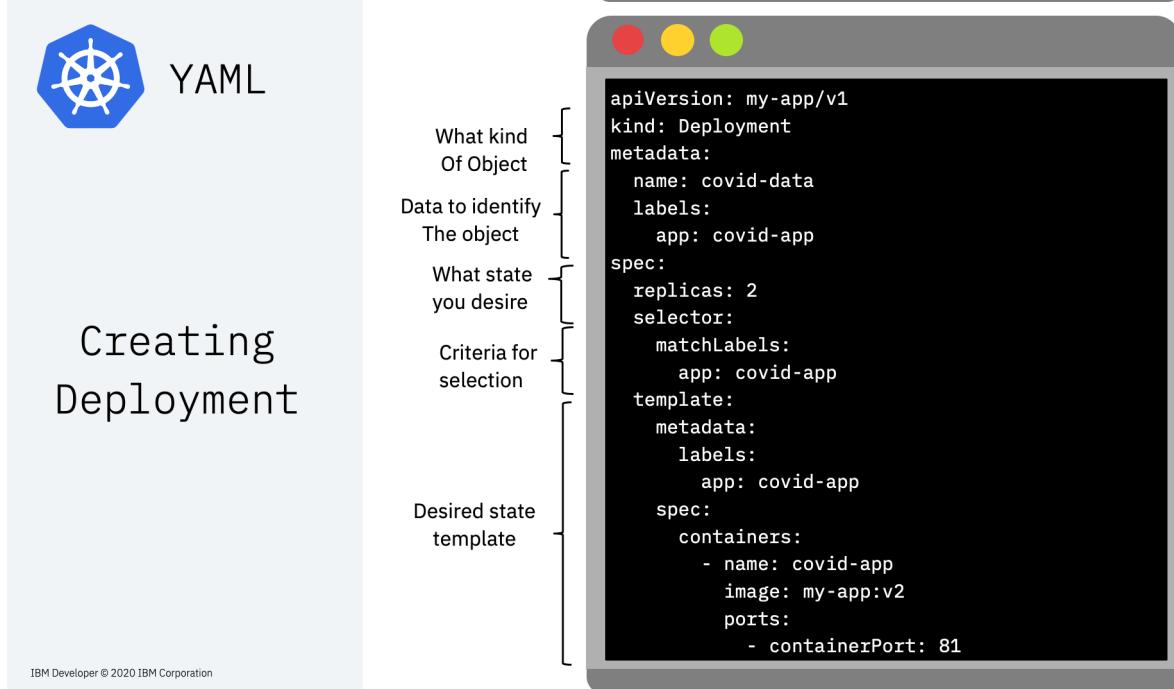
- Print services as Yaml

```
kubectl get services -o yaml
```

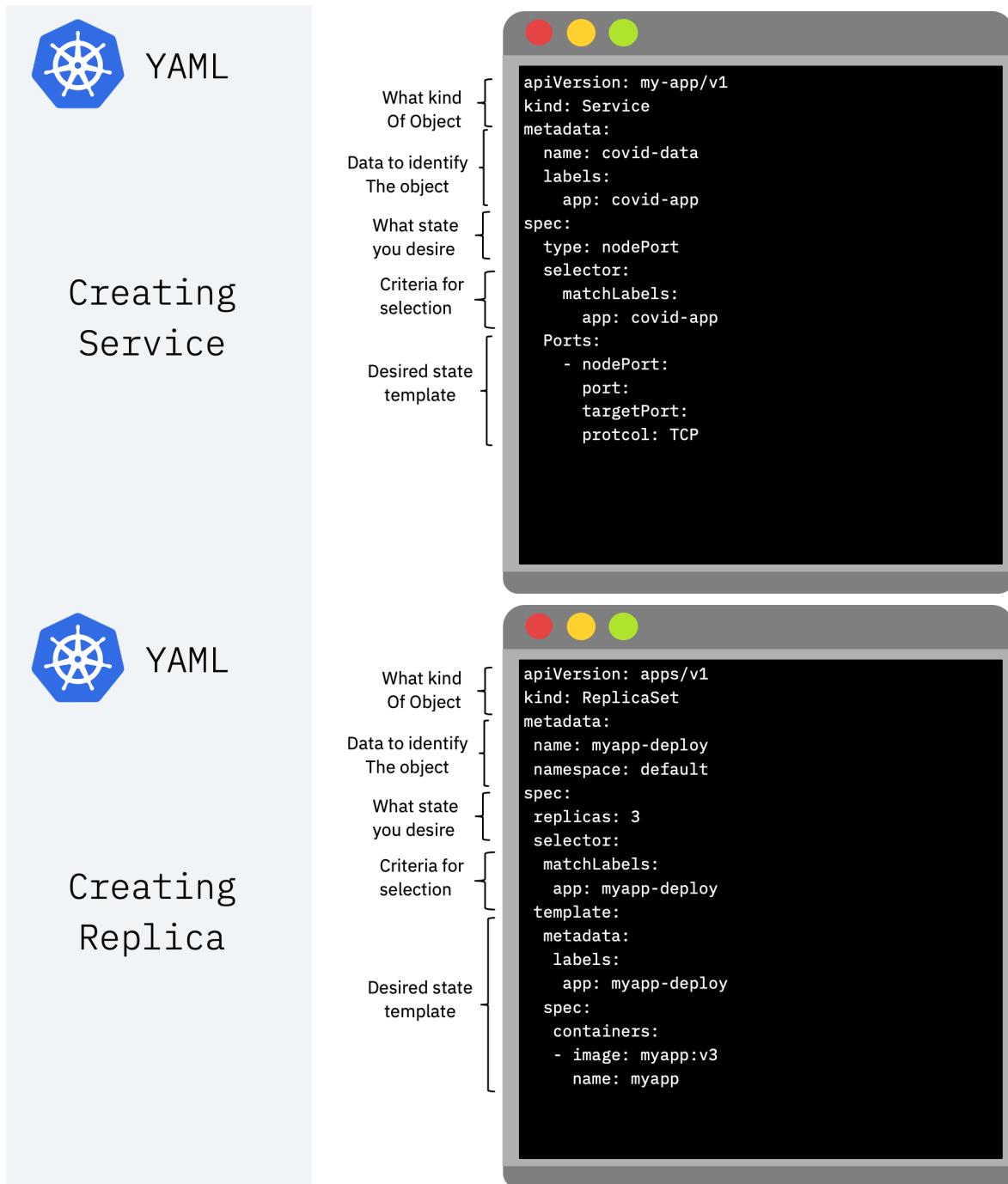
4.22 Using YAML to create resources

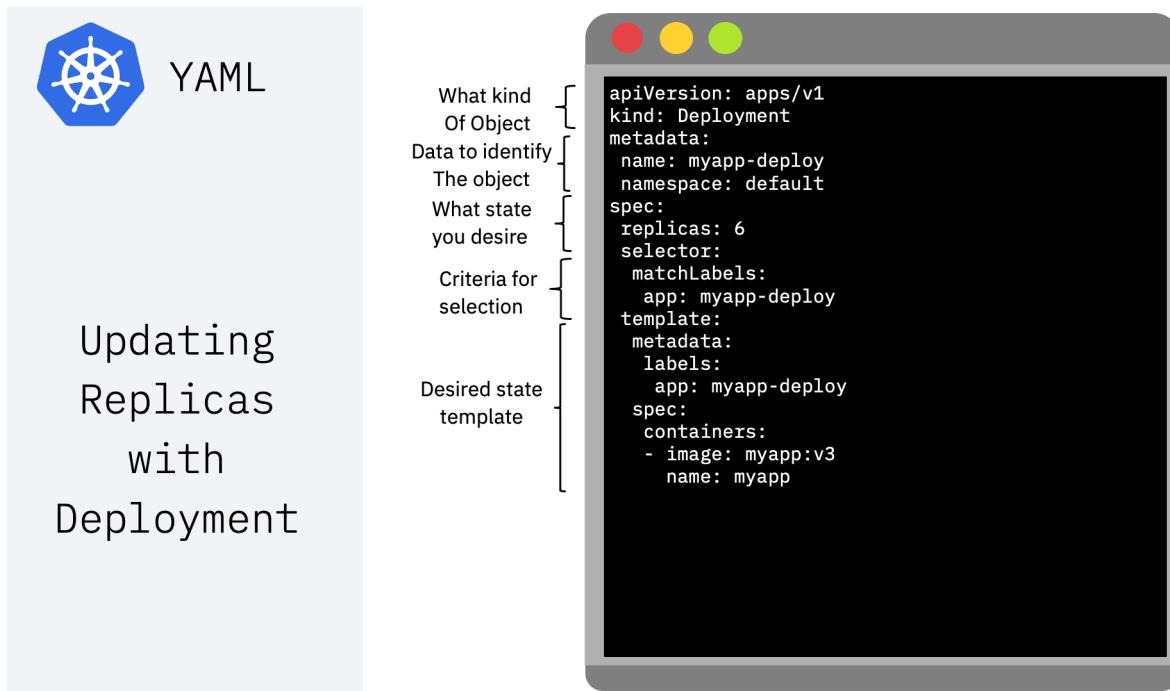


IBM Developer © 2020 IBM Corporation



IBM Developer © 2020 IBM Corporation





4.23 Once YAML file is crafted, here is how to apply it:

```
kubectl apply -f [fileName].yaml
```

4.23.1 Get logs of applying YAML file

```
kubectl log -l app=[container name]
```

4.24 Summary

Summary

Part Three

- Deploy on Kubernetes
 - Minikube
 - Pulling image from registry
 - Create deployment
 - Expose
 - Create services
- Manage with Kubernetes
 - Replicasets
 - Rolling out updates
 - YAML
 - Autoscaling

IBM Developer © 2020 IBM Corporation

@mohaghghi

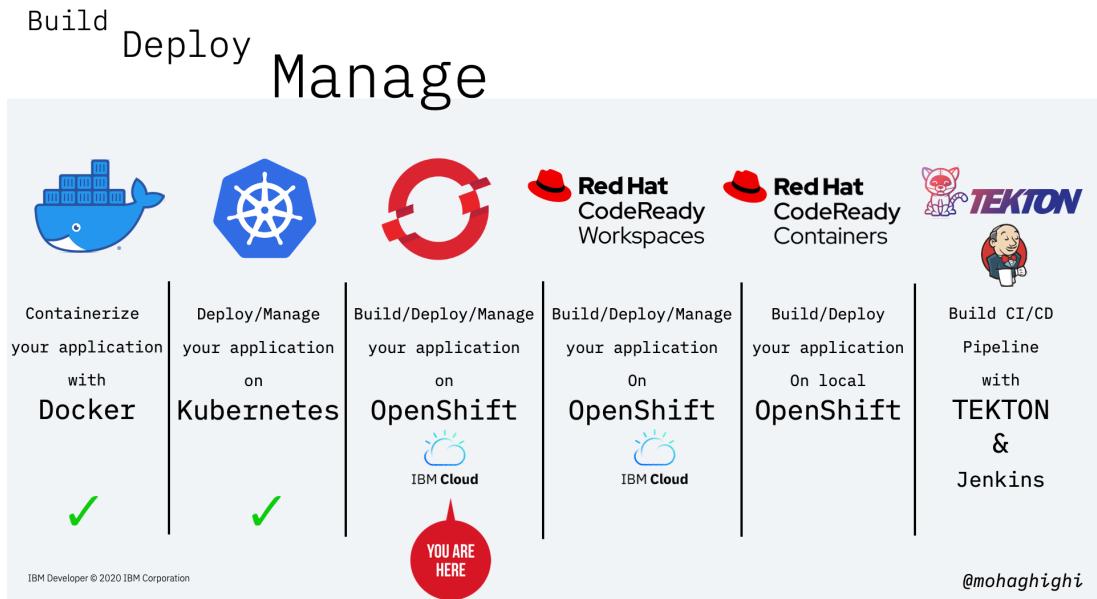
5. Part 4: Build, Deploy and Manage your Microservices Application with OpenShift

The image shows a video player interface. At the top, the title 'Covid-19 Data Analytics with Kubernetes' is displayed. Below it, the speaker's name 'MO HAGHIGHI' and affiliation 'IBM Europe' are shown. A text box on the right side contains the description: 'Step-by-step building, and deploying containerized microservices onto Kubernetes, and managing enterprise distributed applications with OpenShift'. The main title 'Part 4' is centered below the speaker's name. In the bottom right corner, there is a small video frame showing a man speaking. The video player includes standard controls like play, volume, and a progress bar indicating 0:13 / 2:05:45. The IBM Developer logo is visible at the bottom left.

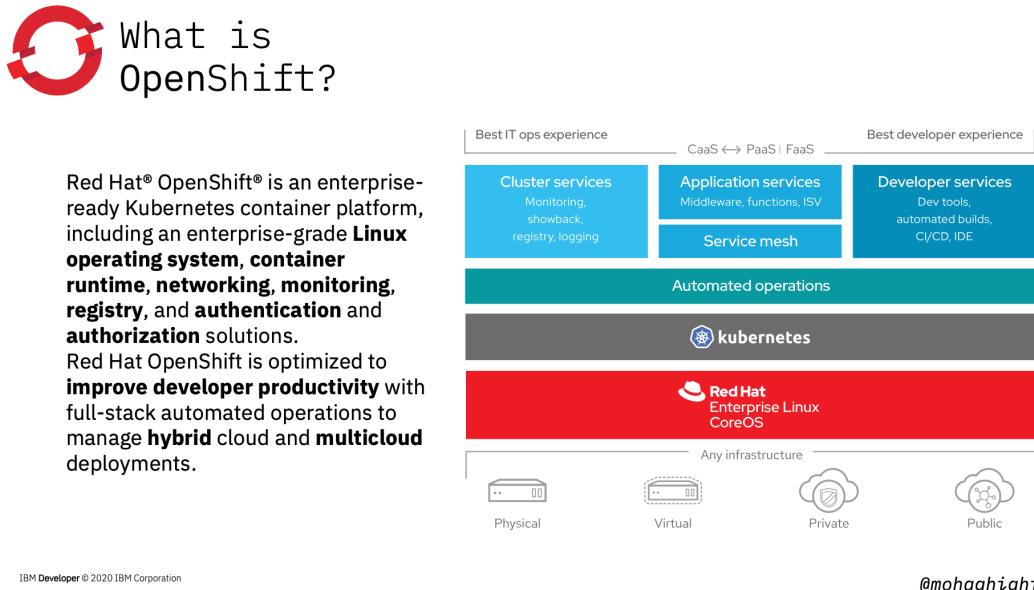
5.1 Agenda

In this section you will learn:

- Why OpenShift?
- Kubernetes vs. OpenShift
- Developer productivity
- Deploy on OpenShift via CLI
- Pushing image to registry
- Create deployment
- Expose
- Deploy on OpenShift via Console
- OpenShift Console
- Builder Images
- S2I (Source to Image)



5.2 What is OpenShift Container Platform?



OpenShift is built on top of Kubernetes, and brings along all the brilliant features of Kubernetes, but it bundles Kubernetes with all the Essential features that will ultimately provide the best experience to both developers and Operation engineers.

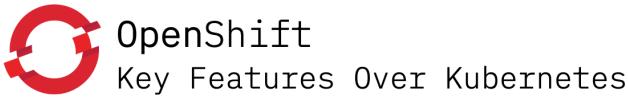
But how does it achieve that?

Through a number of automated workflows, which are not available in Kubernetes.

Those automated workflows are the results of these components that are drawn in this diagram.

Kubernetes is wrapped around an enterprise-grade linux operating system (RHEL/CoreOS), Networking, monitoring, registry, and more importantly, authentication and authorisation.

5.3 3 x key features of OpenShift over Kubernetes. Automation, Agility and Security.



I. Automation

Automated installation, upgrades, and lifecycle management throughout the container stack—the operating system, Kubernetes and cluster services, and applications—**on any cloud**.

II. Agility

Helps teams build with speed, agility, confidence, and choice. **Code in production mode anywhere** you choose to build.

III. Security

Security at every level of the container stack and throughout the application lifecycle.

IBM Developer © 2020 IBM Corporation

@mohaghichi

5.4 what are the automated workflows?



✓ Pre-created quick start application templates to build your application, based on your favourite languages, frameworks, and databases, **with one click**.



✓ Deploying to OpenShift is as easy as clicking a button or entering a **git push** command, **enabling continuous integration**, managing builds, and allows you to fully control the deployment lifecycle.



✓ Develop container-based applications in the cloud or locally using the Red Hat CodeReady Containers to create a fully-functioning OpenShift instance **on your local machine**. Then, deploy your work to any OpenShift cluster.



IBM Developer © 2020 IBM Corporation

@mohaghichi

- As a developer you want to get started on coding as quickly as possible, rather than spending time learning about different platforms, tools and services, and how to refactor your application based on them.

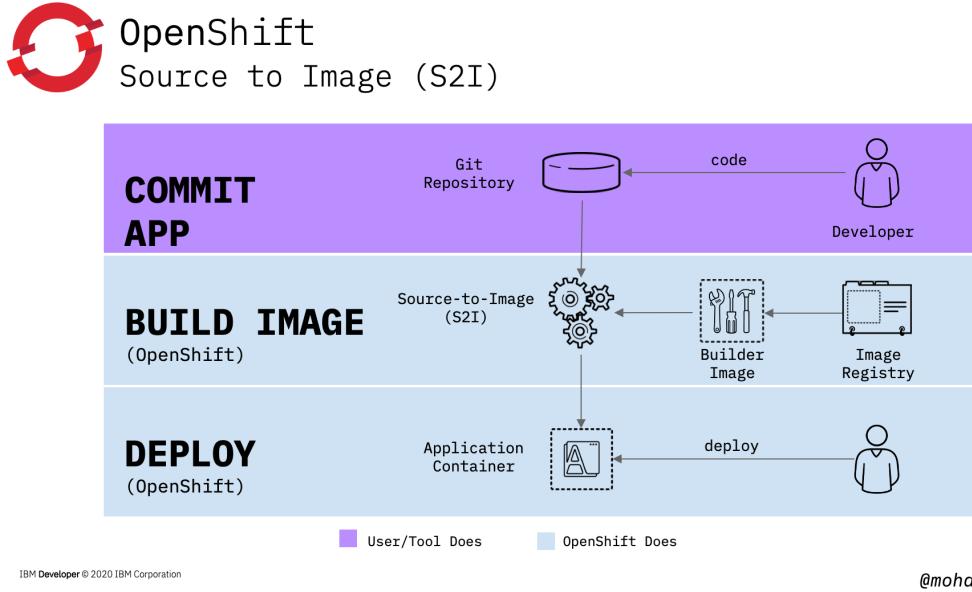
Pre-created quick start application templates to build your application, based on your favourite languages, frameworks, and databases, with one click.

- As a developer you want to focus on coding and not worrying about what's going to happen in the background.

Deploying to OpenShift is as easy as clicking a button or entering a **git push** command, **enabling continuous integration**, managing builds, and allows you to fully control the deployment lifecycle.

- As a developer you want to build and test your application locally, without worrying about the openshift cluster your application will end up running in.

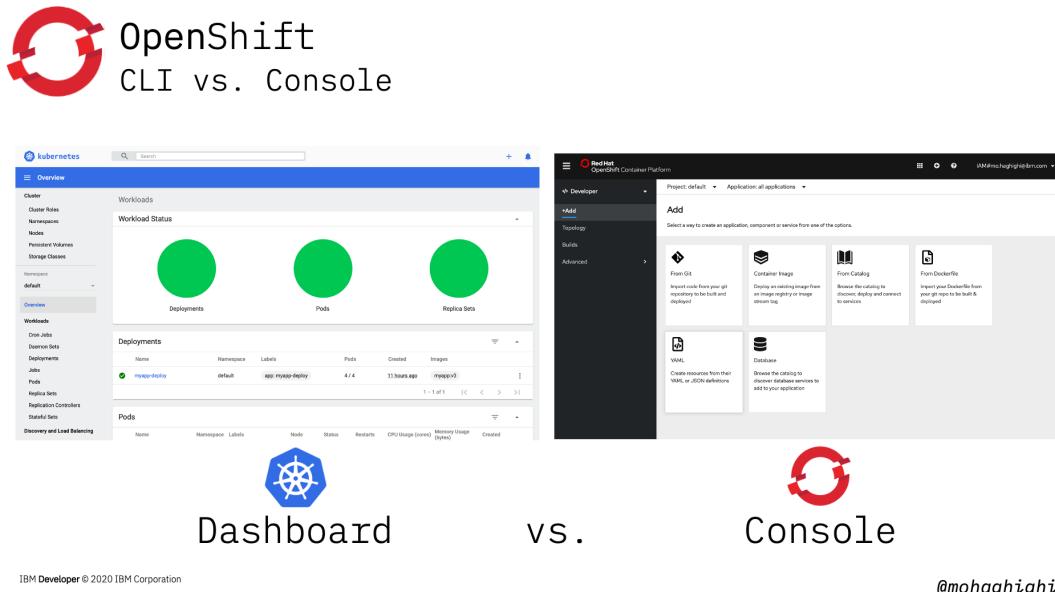
Develop container-based applications in the cloud or locally using the Red Hat CodeReady Containers to create a fully-functioning OpenShift instance **on your local machine**. Then, deploy your work to any OpenShift cluster.



As this figure shows developers can focus on coding, and the rest of the process is taken care of by OpenShift's S2I or Source to Image. Building your image, deploying, and as you will later in part 7, continues integration.

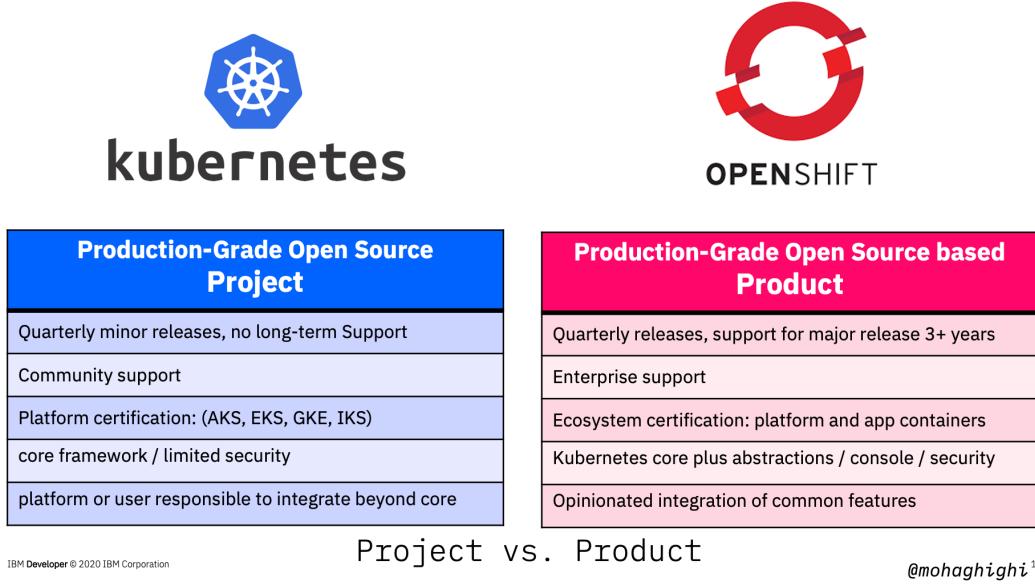
5.5 Three major differences between Kubernetes and OpenShift

5.5.1 CLI vs. Console



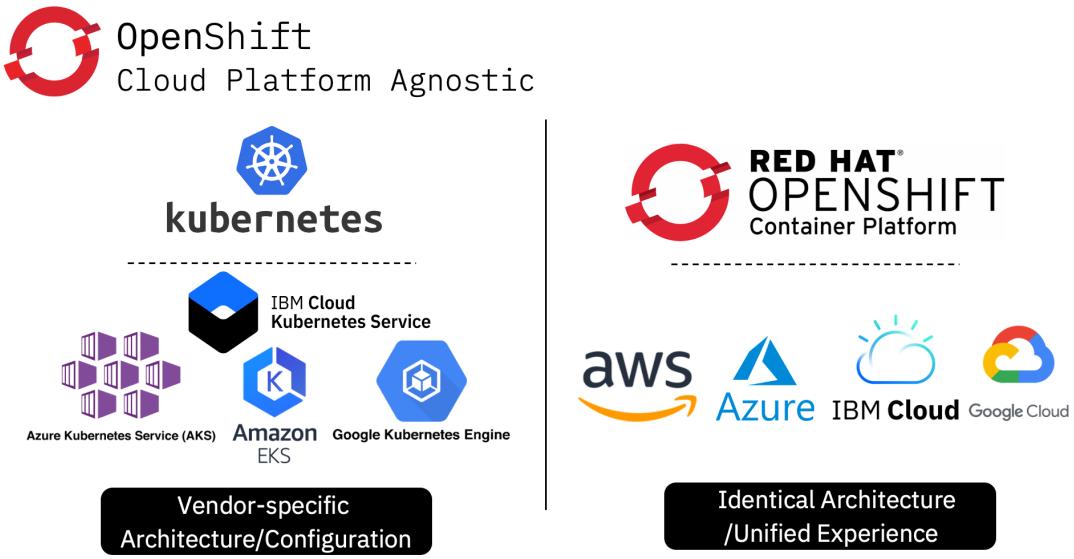
One of the most distinctive features of OpenShift is its amazing web console that allows to implements almost all tasks from a simple graphical interface. As you saw in the previous lab, Kubernetes dashboard is only good for displaying the status of your resources. You can't deploy, control or manage your applications, networking or any of those form Kubernetes dashboard. Obviously, managed Kubernetes on different cloud platforms, come with different set of functionalities as add-ons. But with Openshift container platform, the offered functionalities through the openshift console are vast. You can build, deploy, expose, update, and almost implement any task in two separate perspectives of developer and administrator. We'll go through that later in this lab.

5.5.2 Project vs. Product



Kubernetes is an open source project, whereas OpenShift is a product based on an open source project, which is Kubernetes Origin Distribution or OKD. [next] Comparing Kubernetes with OpenShift is like that classical example of comparing an engine with a car. You can't do much with an engine, and you need to assemble it with other components in order to get from A to B and become productive. What you get with OpenShift includes enterprise support, ecosystem certification, and most importantly, regular releases and security updates at every level of the container stack and throughout the application lifecycle. That is an opinionated integration of features to simplify and secure your applications.

5.5.3 Cloud Platforms Offerings



Kubernetes offerings differ from one platform to another. Almost every major cloud provider offers a different flavour of Kubernetes. You get different sets of add-ons, plug-in and set of instructions for connecting your application to your cloud resources, which in most cases are only applicable to that particular platform. With OpenShift Container Platform, your experience and the way you interact with the platform, let's say the OpenShift console, stays the same. Therefore, building, deploying and managing applications with OpenShift Container Platform is truly: build it once and deploy it anywhere.



OpenShift On IBM Cloud

Red Hat® OpenShift on IBM Cloud™ is a fully managed OpenShift service that leverages the enterprise scale and security of IBM Cloud, so you can focus on growing applications, not scaling the master. IBM has added unique security and productivity capabilities designed to eliminate substantial time spent on updating, scaling and provisioning.

"It's directly integrated into the same Kubernetes service that maintains 25 billion on-demand forecasts daily at The Weather Company"

IBM Developer © 2020 IBM Corporation



@mohaghghi

In this lab we're going to use managed openshift on IBM Cloud. Before continuing, let's get started by provisions an OpenShift cluster on IBM Cloud. Red Hat® OpenShift on IBM Cloud™ is a fully managed OpenShift service that leverages the enterprise scale and security of IBM Cloud, so you can focus on growing applications, not scaling the master.

IBM has added unique security and productivity capabilities designed to eliminate substantial time spent on updating, scaling and provisioning.



OpenShift

Push, Deploy and Expose

```
mo@Ms-MacBook-Pro ~ % oc get pods
NAME                      READY   STATUS    RESTARTS   AGE
codeready-operator-74c6b5df66-kgmqf   1/1    Running   0          2d21h
death-cases-5b6b568b89-lvhkw        1/1    Running   0          6d3h
positive-cases-66c9c9cd99-v2rgh     1/1    Running   0          6d3h
postgres-9c97d6f56-2tqwv           0/1    Pending   0          2d21h
ui-app-5766f874fd-slhwt          1/1    Running   0          6d3h
mo@Ms-MacBook-Pro ~ %
```

Via OC CLI

IBM Developer © 2020 IBM Corporation

@mohaghghi

Once you've signed up on IBM Cloud and sign into your account by visiting cloud.ibm.com, you need to navigate through ibm cloud dashboard and choose OpenShift. Then go ahead and create your cluster. Once your cluster is provisioned and ready, it'll be listed in this table.

5.6 Download and Install prerequisites

Install IBM CLI tools

```
curl -sL https://ibm.biz/1dt-installer | bash
```

Download OC CLI based on local OS and OpenShift version

```
https://mirror.openshift.com/pub/openshift-v4/clients/oc/
```

Download kubectl

```
https://storage.googleapis.com/kubernetes-release/release/v1.17.7/bin/darwin/amd64/kubectl
```

Set your environmental parameters for OC

```
mv <filepath>/oc /usr/local/bin/oc
```

Set your environmental parameters for kubectl

```
mv <filepath>/kubectl/usr/local/bin/kubectl
```

5.7 Login to IBM Cloud and check your installed plugins

Login to IBM Cloud

```
ibmcloud login
```

if using a federated account

```
ibmcloud login --sso
```

List IBM Cloud plugins

```
ibmcloud plugin list
```

List IBM Cloud Openshift clusters

```
ibmcloud oc cluster ls
```

Initialize OC CLI Client

```
ibmcloud oc init
```

Log your local Docker daemon into the IBM Cloud Container Registry

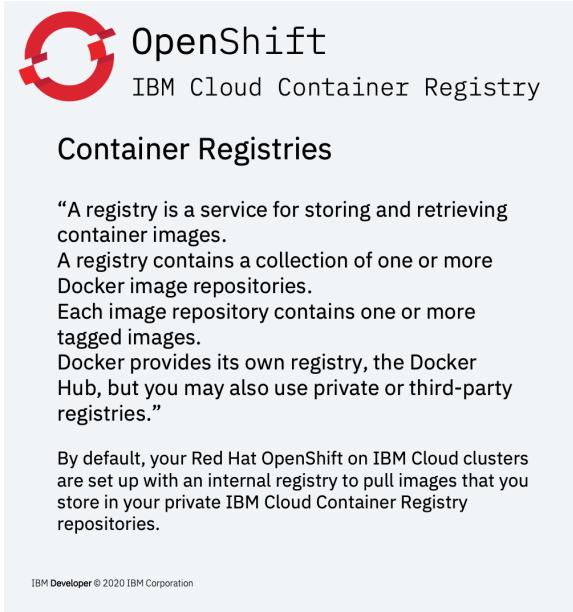
```
ibmcloud cr login
```

Test your OC CLI

```
ibmcloud oc
```

Test your Container Registry

```
ibmcloud cr
```



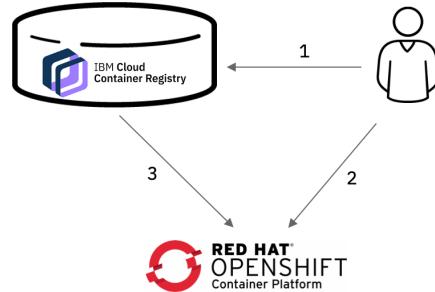
OpenShift
IBM Cloud Container Registry

Container Registries

A registry is a service for storing and retrieving container images.
A registry contains a collection of one or more Docker image repositories.
Each image repository contains one or more tagged images.
Docker provides its own registry, the Docker Hub, but you may also use private or third-party registries.

By default, your Red Hat OpenShift on IBM Cloud clusters are set up with an internal registry to pull images that you store in your private IBM Cloud Container Registry repositories.

IBM Developer © 2020 IBM Corporation

@mohaghghi¹²⁶

5.8 Push Image to IBM Container Registry

Create a new namespace in IBM Cloud Container Registry

```
ibmcloud cr namespace-add [namespace]
```

Tag the image

```
docker tag [image name] us.icr.io/[namespace]/[image name]
```

Push the image to container registry

```
docker push us.icr.io/[namespace]/[image name]
```

List images in IBM Cloud Container Registry

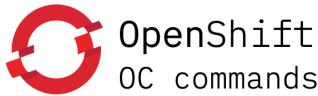
```
ibmcloud cr image-list
```

5.9 OC commands

The developer OC CLI allows interaction with the various objects that are managed by OpenShift Container Platform.

Here is the format of OC commands, almost identical with Kubectl

```
oc <action> <object_type> <object_name>
```



The developer CLI allows interaction with the various objects that are managed by OpenShift Container Platform.

Many common oc operations are invoked using the following syntax:

```
oc <action> <object_type> <object_name>
```

An <action> to perform, such as *get* or *describe*.

The <object_type> to perform the action on, such as *service* or *deployment*

The <object_name> of the specified <object_type>

IBM Developer © 2020 IBM Corporation
@mohaghghi¹²⁸

View existing projects

```
oc projects
```

Switch to a project

```
oc project [project name]
```

Create a new project

```
oc new-project [name project]
```

5.10 Some useful OC commands

Get the full list of OC commands and parameters

```
oc --help
```

In-depth look into the values to be set

```
oc explain [resource]
```

Edit the desired object type

```
oc edit <object_type>/<object_name>
```

Updates one or more fields of an object (The changes is a JSON or YAML expression containing the new fields and the values)

```
oc patch <object_type> <object_name> -p <changes>
```

5.11 Create Deployment using an image from IBM Cloud Container Registry

Create a deployment by instructing the OpenShift cluster to pull an image from ICR

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

Get the list of deployments (same as Kubectl)

```
oc get deployment
```

Get the list of pods (same as Kubectl)

```
oc get pods
```

5.12 Expose the current deployment to the Internet

Expose the deployment on container port 8082 with LoadBalancer service type

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

Get the list of services

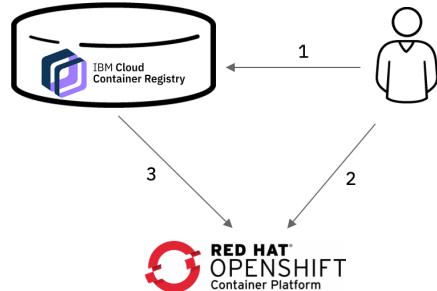
```
oc get services
```



By default, your Red Hat OpenShift on IBM Cloud clusters are set up with an internal registry that stores images locally in your cluster. The clusters are also set up with image pull secrets in the 'default' project to pull images that you store in your private IBM Cloud Container Registry repositories.

OpenShift internal registry to import images from IBM Cloud Container Registry:

- 1 Private registry that is common to multiple clusters
- 2 Images are stored locally on the cluster- reducing latency and external traffic.

IBM Developer © 2020 IBM Corporation
@mohaghghi 131


Every OpenShift project has a Kubernetes service account that is named **default**. Within the project, you can add the image pull secret to this service account to grant access for pods to pull images from your registry.

5.13 Pull Images from ICR into non-Default Projects

- Create an IBM Cloud IAM service ID for your cluster that is used for the IAM policies and API key credentials in the image pull secret.
- Create a custom IBM Cloud IAM policy for your cluster service ID that grants access to IBM Cloud Container Registry.
- Create an API key for the service ID
- Create an image pull secret to store the API key credentials in the cluster project
- Store the registry credentials in a Kubernetes image pull secret and reference this secret from your configuration file.
- Add the image pull secret to your default service account.

Create an IBM Cloud IAM service ID

```
ibmcloud iam service-id-create cluster-project-id --description "service ID for cluster-project"
```

Create a custom IBM Cloud IAM policy for your cluster service ID

```
ibmcloud iam service-policy-create iam-service-id --roles Manager --service-name container-registry
```

Create an API key for the service ID

```
ibmcloud iam service-api-key-create [api-key-name] [service-policy-id] --description "API Key"
```

Create an image pull secret to store the API key & store the registry credentials in K8s image pull secret

```
oc --namespace [project] create secret docker-registry [secret name] --docker-server=us.icr.io --docker-username=iamapikey --docker-password=[API-key] --docker-email=[ ]
```

Get all secrets in project

```
oc get secrets --namespace [project]
```

Get secrets in 'default' serviceaccount in project []

```
oc describe serviceaccount default -n [project]
```

Add the image pull secret to your default service account

```
oc patch -n <project_name> serviceaccount/default --type='json' -p='[{"op":"add","path":"/imagePullSecrets/-","value":{"name":"<image_pull_secret_name>"}]]'
```

Check the secrets again to verify the secret has been added the default serviceaccount.

Get secrets in 'default' serviceaccount in project []

```
oc describe serviceaccount default -n [project]
```

5.14 Verify that the new project can pull images from ICR

Create a deployment by pulling an image from ICR into the new project

```
oc create deployment [new project] --image=us.icr.io/covid-test/myapp:v1
```

verify that image has been pulled and deployed successfully

```
oc get deployment
```

Expose the deployment

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

Verify the service is up and running

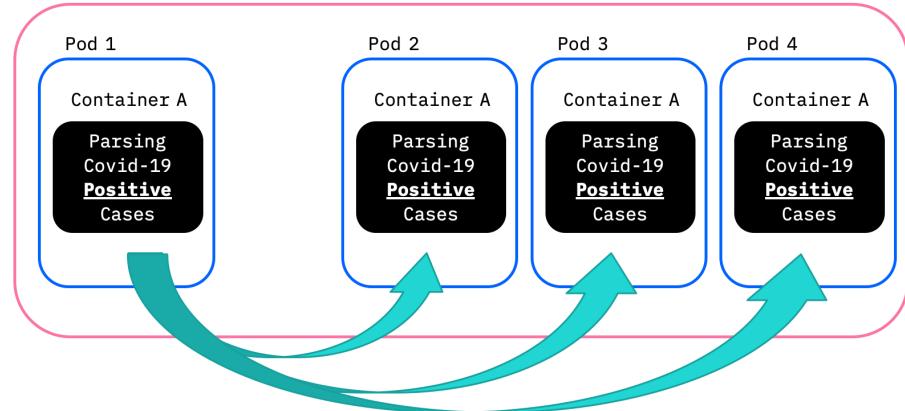
```
oc get services
```

5.15 Scale and Replicas

in this section we will create replicas of our deployed application. Openshift will consider the instructed number of instances as the desired state. If any pod fails or destroyed, OpenShift will bring that back up to keep the number of instances intact in order to meet the load.

OpenShift Scale and Replicas

Worker Node



IBM Developer © 2020 IBM Corporation

@mohaghghi

Scale the application by creating 3 more instances

```
oc scale --replicas=4 deployment/[deployed resource]
```

Get the replicas

```
oc get rs
```

Verify the number of running pods (reflecting the number of instances)

```
oc get pods -o wide
```

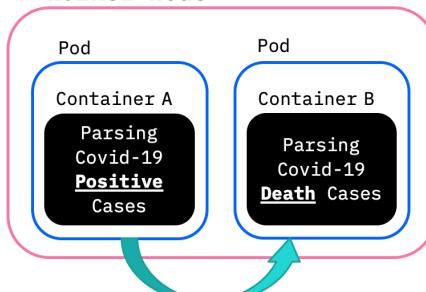
5.16 Rolling out updates and Rolling back

OpenShift Rolling Out Updates

Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. Performing updates without affecting application availability.

- Deploying a new image for parsing Covid-19 Death Rate
- Rolling update by switching to the new image
- updates are versioned and can be reverted to a previous version
- If exposed, service will load-balance the traffic only to available Pods during updating

Worker Node

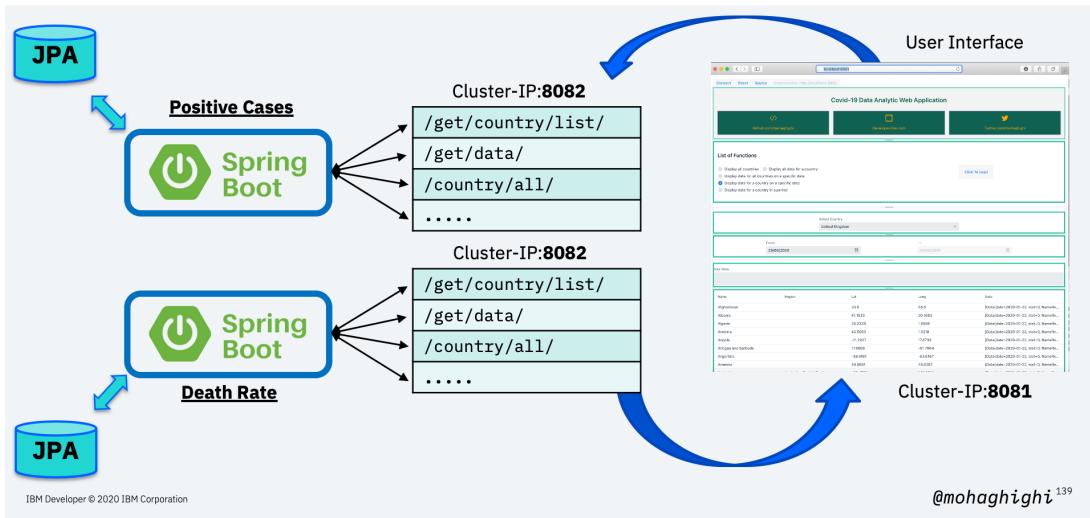


IBM Developer © 2020 IBM Corporation

@mohaghghi

Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. Performing updates without affecting application availability.

Overall Architecture



In this part we're going to update our image to the parser for covid-19 mortality data reflect the number of death in every country country and region.

```
oc set image deployment/[deployment name] [container]=[new image]
```

Make sure you use the container name in the above command to update the image in it.

To get the container name, use:

```
oc get deployment -o wide
```

verify the deployment is updated by pinging the app

```
curl ip:port/hello
curl ip:port/get/country/data/germany/
```

To rollback to the previous version use:

```
oc rollout undo deployment/[deployment Name]
```

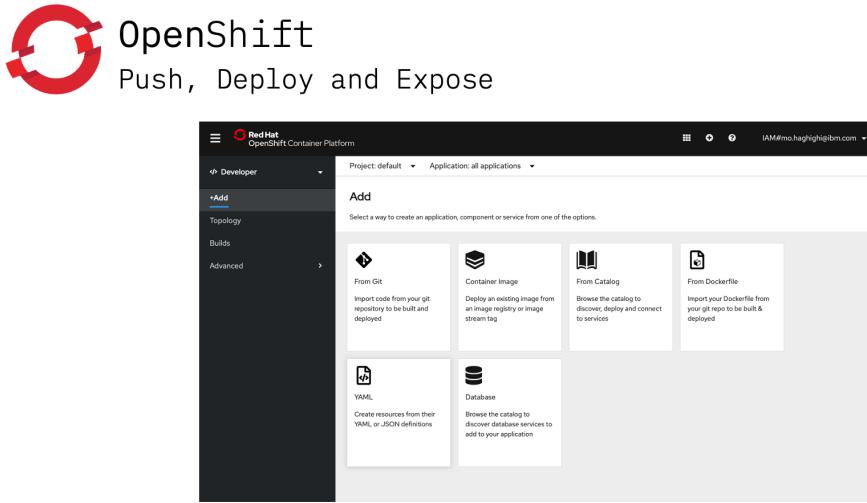
optional: You can add `--to-revision=n` in order to rollback to a specific version

```
oc rollout undo deployment/[deployment Name] --to-revision=2
```

checkout the rollout status

```
oc rollout status deployment/[deployment Name]
```

5.17 Summary



Via Console

IBM Developer © 2020 IBM Corporation

@mohaghighi

6. Part 5: Build, Deploy and Share Your Applications with CodeReady Workspaces

In this lab we'll explore one of the most exciting features of OpenShift for developers. We'll explore how codeready workspaces helps teams build with speed, Agility, security and most notably code: in production from anywhere. And by anywhere, it truly means anywhere as we'll find out shortly.

First We'll take a look at the key features of CodeReady Workspaces and we'll show you how to install code ready workspace in your OpenShift cluster. We'll discuss Operators and the operatorhub. Then we'll dive into our workspace to create a sample application from the in-browser IDE, and share the workspace with our team.

Here's a quick revision of what we've learnt together so far - and how that fits into our learning journey throughout this course. We containerised our application with Docker, deployed and managed with Kubernetes and later with OpenShift CLI and Console. And now we're going to make it even easier to get started with coding from a browser. If you haven't watched the previous workshops, I highly encourage you to go ahead and review them. You get a clear idea about microservices, containerisation, orchestration, how openshift automates tedious tasks, and ultimately why codeready workspaces is such a fabulous solution for developers.

6.1 Agenda

In this section you will learn:

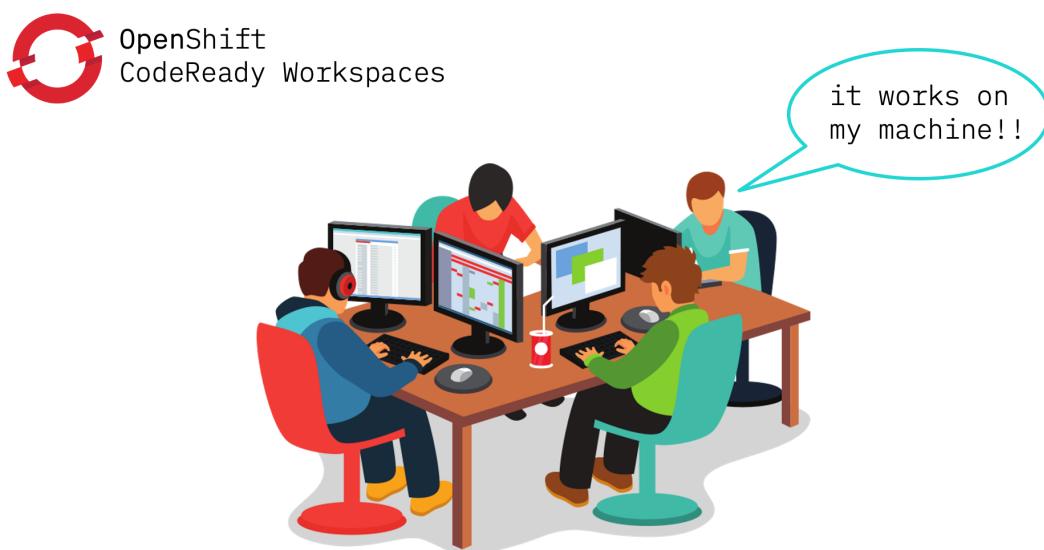
- What is CodeReady workspaces?

- Install CodeReady Workspaces
 - Operators in OpenShift
 - OperatorHub
 - Install CRW Operator
 - Create CheCluster

- Your first workspace
 - Sample stacks
 - Import from Git
 - In-browser IDE
 - Compile/Run/Expose

- Workspace admin

- Share your Workspace

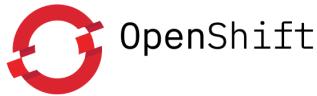


IBM Developer © 2020 IBM Corporation

@mohaghghi 168

Developers often spend too much time configuring their development environment, adding their libraries, dependencies and so forth. It becomes even a bigger problem when developers are collaborating on a project. Let's say you develop an application on your machine, and it runs perfectly. but when others try to run it, all sorts of errors start showing up. And if you're working in a team, despite having kept your team well-aware of all the dependencies and libraries, collaborating on a project becomes a nightmare.

You know that old saying : It works on my machine!!!

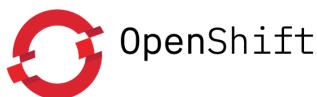


CodeReady Workspaces is a developer tool that makes cloud-native development practical for teams, using Kubernetes and containers to provide any member of the development or IT team with a consistent, preconfigured development environment.

IBM Developer © 2020 IBM Corporation

@mohaghichi

CodeReady workspace offers a shared development environment for rapid cloud application development using Kubernetes and containers to provide a consistent and pre-configured developers environment to your teams.



Allows you to share an instance of your workspace, including all the libraries, dependencies and tools.

Sharing process is as easy as sharing a URL with your team.

Can be accessed from any operating system, browser or IDE, including extension for VS code.

it includes a powerful in-browser IDE, with version control system and keyboard shortcuts.

IBM Developer © 2020 IBM Corporation

@mohaghichi

It is a cloud-native application environment that allows you to share an instance of your workspace, including all the libraries, dependencies and tools.

All you need to do is: add your libraries and dependencies, create a workspace instance and share that with your team members.

It is as easy as sharing a URL - called factory - with the rest of your team. clicking the URL will spin up a new workspace. This way your team will share the same runtime and same development environment.

But that's not all.. CodeReady Workspaces includes a powerful in-browser IDE, with all the features of modern IDEs including version control system and even keyboard shortcuts. You can also access it from any operating system, browser or IDE, including extension for VS code.

The screenshot shows the OpenShift OperatorHub interface. The search bar at the top contains the text 'code'. Below the search bar, there are two columns of operators. The first column contains the 'Joget DX Operator' and the 'No-code/low-code application platform to visually build, run and maintain apps'. The second column contains the 'Red Hat CodeReady Workspaces' operator and the 'vFunction Operator'. The 'Red Hat CodeReady Workspaces' operator is highlighted with a red border. The sidebar on the left shows various categories like Home, Projects, Operators, and Workloads.

@mohaghghi

Installing CodeReady Workspaces in your OpenShift cluster is as simple as looking up its dedicated operator and installing from the OperatorHub within the OpenShift Console.

The screenshot shows the detailed view of the 'Red Hat CodeReady Workspaces' operator. The page includes the operator logo, version (2.2.0), provider type (Red Hat), provider (Red Hat), and repository. A large blue 'Install' button is prominently displayed. To the right of the main content, there is a sidebar with instructions on how to install the operator. The sidebar text reads: 'A collaborative Kubernetes-native development IDE for rapid cloud application development. This operator installs the Red Hat CodeReady Workspaces server, as well as code samples and developer tools.' and 'How to Install'. The sidebar also provides steps for upgrading and creating a new CRD. The sidebar is highlighted with a red border.

@mohaghghi

Now let's explore Operators and the OperatorHub:

The screenshot shows the OpenShift Container Platform interface. In the top left, there's a logo for 'Red Hat OpenShift Container Platform'. The sidebar on the left is titled 'Administrator' and includes links for Home, Dashboards, Projects, Search, Explore, and Events. The main content area has a header 'Project: default' and a sub-header 'Installed Operators'. Below this is a table with columns: Name, Namespace, Deployment, and Status. One row in the table is highlighted with a pink box, showing 'Red Hat CodeReady Workspaces' in the Name column, 'default' in the Namespace column, 'Red Hat CodeReady operator' in the Deployment column, and 'Succeeded Up to date' in the Status column. At the bottom of the page, it says 'IBM Developer © 2020 IBM Corporation'.

Operators

Operators are small programs in your cluster that monitor your applications continuously and make sure they are running according to your instructions. When an operator detects a difference between the actual and the ideal states, it will act to correct it.

OperatorHub

A catalogue of applications/operators that can be installed by the administrator and added to individual projects by developers in OpenShift 4.

@mohaghghi

6.2 what is an operator?

updating and maintaining containerised applications should be an automated process. The same applies to your containerised development environment. Operators are small programs in your cluster that monitor your applications continuously and make sure they are running according to your instructions. When an operator detects a difference between the actual and the ideal states, it will act to correct it.

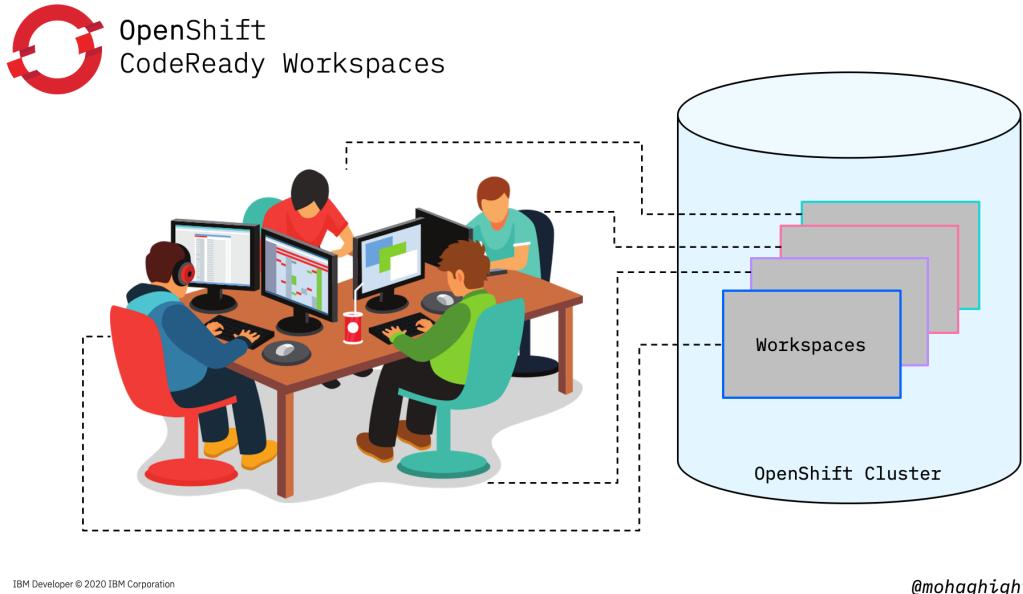
If you recall from workshop 3, we discussed how Kubernetes master node continuously reconciles the expressed desired state and the current state of an object. And that is a controller in Kubernetes. Controller is a core concept in Kubernetes and is implemented as a software loop that runs continuously on the Kubernetes master node.

An Operator is essentially a custom controller.

The Operator is a piece of software running in a Pod on the cluster, interacting with the Kubernetes API server.

6.3 What is the OperatorHub:

Operators are offered as pre-packaged modules from the operatorhub. OpenShift 4 introduced the OperatorHub, and that is a catalog of applications that can be installed by the administrator and added to individual projects by developers.



IBM Developer © 2020 IBM Corporation

@mohaghghi 174

As we mentioned, Codeready workspaces is offered as a dedicated operator from the openshift Operatorhub.

Regardless of where you have your open shift cluster running, Codeready workspace runs as a pod inside your cluster.

Therefore workspaces are maintained and updated by an operator and you can rest assured that your development environment is always available and running according to your requirement.

Underneath each workspace is a stack, a container image that includes language runtimes, compilers, tools, and utilities. Red Hat CodeReady Workspaces ships with stacks for many different languages. Stacks can go beyond just language support, however. A stack can contain multiple containers, allowing you to code in a replica of your production environment.

6.4 Install CodeReady Workspaces

The screenshot shows the 'Create Operator Subscription' page for the Red Hat CodeReady Workspaces operator. The left sidebar is titled 'Administrator' and has sections for Home, Dashboards, Projects, Search, Explore, Events, Operators (selected), OperatorHub, Installed Operators, Workloads, Networking, Storage, Builds, and Monitoring. The main content area is titled 'Create Operator Subscription' and includes fields for 'Installation Mode' (set to 'A specific namespace on the cluster'), 'Update Channel' (set to 'latest'), and 'Approval Strategy' (set to 'Automatic'). A 'Provided APIs' section lists 'Red Hat CodeReady Workspaces' and 'CodeReady Workspaces Cluster'. At the bottom are 'Subscribe' and 'Cancel' buttons.

IBM Developer © 2020 IBM Corporation

@mohaghichi

The screenshot shows the 'Installed Operators' page. The left sidebar is identical to the previous one. The main content area shows a table of installed operators, with the 'Red Hat CodeReady Workspaces' operator listed. It has a status of 'Succeeded' and 'Up to date'. The 'Provided APIs' column shows 'CodeReady Workspaces Cluster'. A pink box highlights the 'Installed Operators' section in the sidebar and the operator entry in the table.

IBM Developer © 2020 IBM Corporation

@mohaghichi

The screenshot shows the 'Operator Details' page for the Red Hat CodeReady Workspaces operator. The left sidebar is identical. The main content area shows the 'Overview' tab selected. It displays the operator's name ('Red Hat CodeReady Workspaces'), version ('2.2.0'), provider ('Red Hat'), and a 'Create Instance' button. Below this is a 'Provided APIs' section for 'CodeReady Workspaces Cluster'. On the right side, there are detailed sections for 'Description', 'How to Install', and 'Actions' (which includes links to GitHub, documentation, and maintainers).

IBM Developer © 2020 IBM Corporation

@mohaghichi

The screenshot shows the OpenShift CodeReady Workspaces interface. The left sidebar is titled 'Administrator' and includes sections for Home, Operators (selected), OperatorHub, Installed Operators, Workloads, Networking, Storage, Builds, Monitoring, Compute, and User Management. The main content area is titled 'Project: test' and shows the 'Red Hat CodeReady Workspaces' operator details. The 'CodeReady Workspaces Cluster' tab is selected. Below it, there's a section titled 'CheClusters' with a 'Create CheCluster' button highlighted by a pink box. A search bar labeled 'Filter by name...' is also visible.

@mohaghghi

This screenshot shows the 'Create CheCluster' form. It contains YAML code for defining a CheCluster, with lines 1 through 26 visible. At the bottom of the form, there are 'Create' and 'Cancel' buttons, with the 'Create' button highlighted by a pink box. To the right, there's a 'View shortcuts' section and a detailed description of the 'CheCluster' schema.

@mohaghghi

The screenshot shows the 'CodeReady Workspaces Cluster' list page. It displays a table with columns: Name, Labels, Kind, Status, Version, and Last Updated. There is one entry: 'codeready-workspaces' with 'No labels', 'CheCluster' as the kind, 'Unknown' status, 'Unknown' version, and 'a minute ago' last updated. The 'Name' column is sorted in ascending order. A pink box highlights the 'codeready-workspaces' entry.

@mohaghghi

The screenshot shows the OpenShift CodeReady Workspaces interface. At the top, there's a navigation bar with the Red Hat logo and the text "OpenShift Container Platform". Below it is a toolbar with filters: Deployment (5), Service (5), ReplicaSet (6), Pod (5), Secret (0), ConfigMap (2), and "Select All Filters". A pink box highlights the "Pod" section of the resource list.

Name	Kind	Status	Created
devfile-registry-6785b4f967-kbkm	Pod	Running	less than a minute ago
keycloak-8bf578486-5zrjm	Pod	Running	a minute ago
plugin-registry-694d4d6b76-fbfzs	Pod	Running	a few seconds ago
postgres-f56d449f6-9kj4c	Pod	Running	4 minutes ago
codeready-7974bfd8b6-sn6k5	Pod	Pending	Jul 13, 8:13 am
che	ConfigMap	Created	a few seconds ago
che-host	Service	Created	2 minutes ago

At the bottom left, it says "IBM Developer © 2020 IBM Corporation". On the right, there's a watermark "@mohaghichi".

The screenshot shows the "CodeReady Workspaces Cluster Overview" page. On the left, there's a sidebar with navigation links like Home, Operators, Workloads, Networking, Storage, Builds, Monitoring, Compute, User Management, and Administrators. The "Operators" link is highlighted.

The main content area has tabs for "Overview" (which is selected), "YAML", and "Resources". The "Overview" tab displays the following information:

- Name:** codeready-workspaces
- Namespace:** NS test
- Labels:** No labels
- Annotations:** 0 Annotations
- Created At:** 8 minutes ago
- Owner:** No owner
- Status:** Available
- OpenShift OAuth:** False
- TLS Mode:** True
- CodeReady Workspaces URL:** codeready-test.myoc43-284105la6073aca7cf26665e142edfa-0000.us-south.containers.appdomain.cloud
- Red Hat SSO Admin Console URL:** keycloak-test.myoc43-284105la6073aca7cf26665e142edfa-0000.us-south.containers.appdomain.cloud
- CodeReady Workspaces version:** 2.2

At the bottom left, it says "IBM Developer © 2020 IBM Corporation". On the right, there's a watermark "@mohaghichi".

The screenshot shows a registration form titled "Register". The form fields are:

- First name
- Last name
- Email
- Username
- Password
- Confirm password

Below the form is a link "[« Back to Login](#)". At the bottom is a blue "Register" button. The background features a dark, geometric pattern.

At the bottom left, it says "IBM Developer © 2020 IBM Corporation". On the right, there's a watermark "@mohaghichi".

The screenshot shows the 'Getting Started with CodeReady Workspaces' page. On the left is a sidebar with navigation options: 'Workspaces', 'Get Started' (which is selected), 'Stacks', 'Factories', and 'Administration'. Below this are sections for 'RECENT WORKSPACES' and 'DESKTOP APPS'. The main area displays a grid of workspace samples:

- Java EAP Maven**: Java stack with EAP 7.3, OpenJDK 8 and Maven 3.5.
- Java EAP Thorntail**: Quickstart to expose a REST Greeting endpoint using Thorntail with EAP 7.2.7, OpenJDK 8 and Maven 3.6.3.
- Red Hat Fuse**: Red Hat Fuse stack with OpenJDK 8 and Maven 3.6.3.
- Java Maven**: Java stack with OpenJDK 11, Maven 3.6.3 and Vert.x demo application.
- Apache Camel K**: Tooling to develop Integration projects with Apache Camel K.
- Java Gradle**: Java stack with OpenJDK 11, Gradle 6.1 and Maven 3.6.3.
- QUARKUS**: Java Quarkus stack with Quarkus Tools, Gradle 6.1 and Maven 3.6.3.
- Java Spring Boot**: Java stack with OpenJDK 8, Maven 3.6.3 and Spring Boot Petclinic demo application.

At the bottom left is the footer: 'IBM Developer © 2020 IBM Corporation'. At the bottom right is the handle '@mohaghhighi'.

The screenshot shows the 'Create Workspace' page. The sidebar includes 'Workspaces (1)', 'Get Started', 'Stacks', 'Factories', and 'Administration'. The main area has a 'Create Workspace' button and a 'Recent Workspaces' section. A progress bar at the bottom indicates 'Loading...'. The terminal output shows the creation process:

```

Processing plugin redhat/aval/latest
Installing plugin extension 1/2
  From https://plugins.redhat.com/v3/resources/download/jboss.org/vscode/3rdparty/vscode-java-debug-vscode-0.24.0.vsix
  into /home/mohaghhighi/.vscode/extensions/redhat.vscode-0.24.0
  Downloading plugin from https://plugins.redhat.com/v3/resources/download/jboss.org/vscode/3rdparty/vscode-java-debug-vscode-0.24.0.vsix
Processing plugin redhat/dependency-analytic/latent
  From https://plugins.redhat.com/v3/resources/download/jboss.org/jneustadt/extension/java-dependency-analytic/0.1.0-2137.vsix
  into /home/mohaghhighi/.vscode/extensions/redhat.java-dependency-analytic-0.1.0-2137
  Downloading plugin from https://plugins.redhat.com/v3/resources/download/jboss.org/vscode/3rdparty/vscode-javascript-0.11.0.vsix
  Saving log of installed plugin
  All 2 plugins have been successfully downloaded
  Pulling Image registry.redhat.io/codeready-workspaces/wiproxy-redhat8:227e0608f75ab7e2fd801affcd93c5a7edc393845b36701a
  Created container che-wiproxyhold008
  Starting container che-wiproxyhold008
  Pulling Image registry.redhat.io/codeready-workspaces/plugin-java8:che18@sha256:6d3781f527e0608f75ab7e2fd801affcd93c5a7edc393845b36701a

```

At the bottom left is the footer: 'IBM Developer © 2020 IBM Corporation'. At the bottom right is the handle '@mohaghhighi'.

The screenshot shows the Red Hat CodeReady Workspaces desktop application. The window title is 'Red Hat CodeReady Workspaces'. The main area displays a 'Welcome To Your Workspace' message. A terminal window is open with the command 'Repository URL' highlighted. A context menu is open in the top-left corner with options like 'New File...', 'Open...', and 'Git Clone...'. The bottom status bar shows 'Che Workspace: Finished importing projects.' and 'Che Workspace: Starting importing projects.' At the bottom left is the footer: 'IBM Developer © 2020 IBM Corporation'. At the bottom right is the handle '@mohaghhighi'.

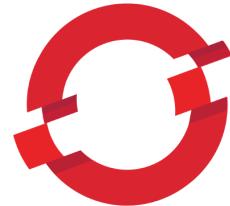
6.5 Summary

Summary

Part Five

- What is CodeReady workspaces?
- Install CodeReady Workspaces
 - Operators in OpenShift
 - OperatorHub
 - Install CRW Operator
 - Create CheCluster
- Your first workspace
 - Sample stacks
 - Import from Git
 - In-browser IDE
 - Compile/Run/Expose
- Workspace admin
- Share your Workspace

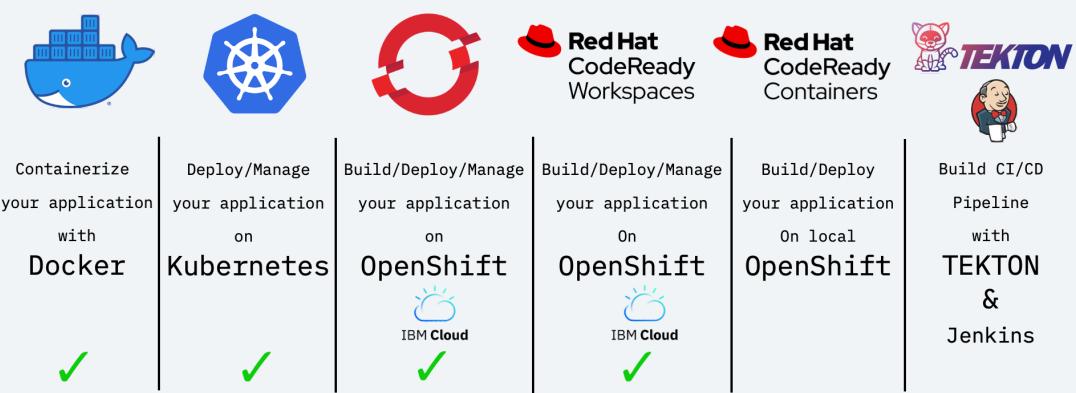
IBM Developer © 2020 IBM Corporation



Red Hat
CodeReady
Workspaces

@mohaghghi

Build Deploy Manage



IBM Developer © 2020 IBM Corporation

@mohaghghi

7. Part 6: Build, and Test Your Applications with CodeReady Containers



CodeReady Containers brings a minimal, preconfigured OpenShift 4.1 or newer cluster to your local laptop or desktop computer for development and testing purposes.

IBM Developer © 2020 IBM Corporation
@mohaghghi

CodeReady Containers brings a minimal, preconfigured OpenShift 4.x to your local laptop or desktop computer for development and testing purposes. CodeReady Containers is delivered as a Red Hat Enterprise Linux virtual machine that supports native hypervisors for Linux, macOS, and Windows 10.

CodeReady Containers is the quickest way to get started building OpenShift clusters. It is designed to run on a local computer to simplify setup and testing, and emulate the cloud development environment locally with all the tools needed to develop container-based apps.

7.1 Agenda

In this section you will learn:

- What is CodeReady Containers?
 - Install & Setup
 - Start CodeReady Containers
- Build on CodeReady Containers
 - From Git
 - From Templates
 - From Containers
 - From Dockerfile
- Deploy with Source to Image from the console
- View our resources from the CLI

Download CodeReady Containers (CRC) from this link after signing up for a Red Hat Developer account.

Prerequisites

	https://spring.io/guides/gs/spring-boot/	2.2
	https://openjdk.java.net/install/	8/11
	https://netbeans.apache.org/download/	12
	https://nodejs.org/en/download/	14
	https://docs.docker.com/engine/install/	
	https://kubernetes.io/docs/tasks/tools/install-minikube/	
	https://developers.redhat.com/products/codeready-containers	
	https://www.ibm.com/cloud/openshift	

@mohaghghi¹⁹⁴

Once CRC is downloaded, set it up by following these commands:

```
crc setup
```

Then start your CRC:

```
crc start
```

You will be asked to enter your **pull secret**. Retrieve it from your Red Hat account:

OpenShift
Setup and Run CRC

99. `crc setup`
100. `crc start`
101. `eval $(crc oc-env)`
102. `crc status`

Downloads

Download and extract the CodeReady Containers archive for your operating system to set up your host operating system for the CodeReady Containers virtual machine.

- Windows: [Download \(HyperV\)](#)
Note: Only supported on Windows 10 Pro or Home with the Fall Creator's Update. No other version or edition of Windows is supported at this time.
- macOS: [Download \(HyperKit\)](#)
- Linux: [Download \(Libvirt\)](#)

Pull secret

Download or copy your pull secret. The install program will prompt you for your pull secret.

[Download pull secret](#) [Copy pull secret](#)

Note: Your CodeReady Container cluster will not show in your list of clusters in OpenShift Cluster Manager currently shows only production-level cluster.

```
tar -xzf [file].tar.xz
```

```
mv ./crc /usr/local/bin/crc
```

IBM Developer © 2020 IBM Corporation

@mohaghghi

Once CRC starts, you will be provided with dedicated URLs to log into your CRC webconsole as an admin or developer:

```

INFO Checking size of the disk image /Users/mo/.crc/cache/crc_hyperkit_4.4.8/crc.qcow2 ...
INFO Creating CodeReady Containers VM for OpenShift 4.4.8...
INFO CodeReady Containers VM is running
INFO Verifying validity of the cluster certificates ...
INFO Restoring the host network ...
INFO Check internal and public DNS query ...
INFO Check DNS query from host ...
INFO Generating new SSH key ...
INFO Copying kubeconfig file to instance dir ...
INFO Starting OpenShift kubelet service
INFO Configuring cluster for first start
INFO Adding user's pull secret ...
y running 'oc login -u developer -p developer https://api.crc.testing:6443'
n 'oc login -u kubeadmin -p fq66o-KsVBU-cnKBU-xLpqd https://api.crc.testing:6443'

INFO You can now run 'crc console' and use these credentials to access the OpenShift web console
Started the OpenShift cluster
WARN The cluster might report a degraded or error state. This is expected since several operators have been disabled to lower the resource usage. For more information, please consult the documentation
mo@Mo-MacBook-Pro ~ %

```

IBM Developer © 2020 IBM Corporation

@mohaghghi

You will need the username and password in order to log into the web console.

IBM Developer © 2020 IBM Corporation

@mohaghghi

If you want to carry on using the CLI tool, make sure you've set your environmental parameters to interact with CRC using OC commands:

```
eval $(crc oc-env)
```

some extra options to include in your CRC:

You can define your allocated resources by adding options to control the number of CPU cores, memory and the Hypervisor used by CRC

```
crc start --cpus [cpu cores] --memory [mib] --vm-driver [vm]
```

by default CRC loads this way

```
crc start --cpus [4] --memory [8192] --vm-driver [hyperkit]
```

To stop CRC

```
crc stop
```

8. Part 7: Build your CI/CD pipelines with Jenkins and Tekton

8.1 Agenda

In this section you will learn:

- Install/download prerequisites
- Package Java Maven application
- Test Java application
- Docker
 - Dockerfile
 - Build Docker image
 - Run Docker containers
 - Use Kubernetes Docker daemon
 - Docker Registry
 - SSH into Docker images
 - Connecting Docker containers
 - Inspect Docker Containers