EPAM Cloud Orchestrator

Containerization Service

User Guide

June 2018

CSUG

Version 1.3

CONTENTS

Preface 4
About this Guide 4
Audience 4
Structure of the Guide 4
Documentation References 4
1 Containerization as a Service 5
2 Containerization Basics 5
3 Container Management Tools 5
3.1 Docker Service 5
3.1.1 Working with Containers 6
3.1.2 Working with Docker Images via Docker Registry 7
3.1.3 Working with Docker Volumes 10
3.1.4 Docker Info 11
3.1.5 REST API 12
3.1.6 Pricing 12
3.2 Kubernetes Service 14
3.2.1 Service Activation 14
3.2.2 Kubernetes Info 15
3.2.3 Kubernetes Pods Management 15
3.2.4 Kubernetes Namespaces Management 15
3.2.5 Kubernetes Services Management 16
3.2.6 Kubernetes Replication Controllers Management 16
3.2.7 Kubernetes Nodes Management 16
3.2.8 Using the Service 17
3.2.9 kubectl Configuration 17
3.2.10 RBAC Authorization 18
3.2.11 Web UI 19
3.2.12 Container Images 19
3.2.13 Troubleshooting 20
3.2.14 Pricing 20

Table of Figures 21

Version History 22

PREFACE

ABOUT THIS GUIDE

This document describes the containerization platforms and the related services implemented in EPAM Cloud. It contains the basic guidelines on deploying and using containerization services in virtual infrastructures.

AUDIENCE

This guide is designed for Cloud users who use containerized infrastructure in their projects.

STRUCTURE OF THE GUIDE

The guide consists of the following sections:

1. Containerization as a Service. This section provides general overview of the containerization service implemented in EPAM Cloud.
2. Containerization Basics. This section explains the components of containerization service and their interaction.
3. Container Management Tools. This section contains a brief description of Docker Swarm and Kubernetes clustering platforms and their comparison. The subsections give detailed instruction on activating and using the services.

DOCUMENTATION REFERENCES

EPAM Orchestration is described in details in a number of documents, oriented on different aspects of Orchestration usage, and on different types of users.

You can find these documents on our Documentation page.

The answers to the most frequently asked questions can be found on the FAQ page.

EPAM Cloud terms and conditions are described in the Terms and Conditions. Please take a look at this document in order to avoid misunderstandings and conflicts that may arise during the service usage.

The terminology of EPAM Cloud and the related products can be found on the Glossary page.

Please email your comments and feedback to EPAM Cloud Consulting at

SpecialEPM-CSUPConsulting@epam.com to help us provide you with documentation that is as clear, correct and readable as possible.

1 CONTAINERIZATION AS A SERVICE

Containerization is a concept of deploying applications in virtual infrastructure where all components of each application are held within a single isolated, independent “container”. Containerization has multiple benefits over the traditional deployment – faster and more efficient installation, environment consistency, portability, security. Additionally, containerized deployment is more economic, as multiple containers share the resources of the same host OS.

EPAM Cloud offers containerization based on Docker, an open-source software for quick and efficient container creation. Currently, Docker 1.8 version is supported.

2 CONTAINERIZATION BASICS

Generally, a Docker container is a virtual Linux-based server on which an application is deployed. To run new containers, Docker uses **images**, that is, sets of software components which should be installed for the desired application deployment. Docker images can be created from scratch, cloned from existing containers or obtained from Docker registries.

Containerized applications run on **clusters** of virtual machines. Clusters consist of a **master** VM from which the environment is managed, and one or more **nodes** on which containers are deployed and running.

3 CONTAINER MANAGEMENT TOOLS

Container management is implemented through Docker Swarm (version 1.0.0), a native Docker clustering tool, or Kubernetes (version 1.7.15), an open-source container management platform delivered by Google, used to manage, schedule and run containerized applications. Docker Swarm, being a native Docker tool, has maximum compatibility with Docker and similar management. On the other hand, the most common Docker limitations will affect Docker Swarm as well. Kubernetes, as a Google platform, has different controls and management system but can perform better where the Swarm functionality is limited. Docker Swarm is the optimal choice when the number of containers is relatively small, while Kubernetes is more efficient with large numbers of containers.

For more details on Docker and Kubernetes and their advantages please visit the Official Docker Website and Official Kubernetes Website.

3.1 DOCKER SERVICE

Docker is the base platform on which containerized infrastructure is built. The Docker service manages the following entities:

- **Master** – a Docker host with a Docker manager (Swarm) that performs Docker cluster health check, load balancing on containers creation, and collects information on existing images, containers, configuration, statuses etc.
- **Node** – a VM that plays a role of a base for containers. A Node VM is created by Orchestrator on Docker Service activation

- **Container** – a node-hosted resource that uses a part of node capacities and can be used as a typical virtual server.
- **Container Image** – an image stored on the node VM and used for containers creation
- **Volume** – a container directory mapped to a host directory and used to store and share data
- **Docker Registry** – a repository service that allows you to share VM images between nodes. Registry is hosted on a separate VM that is used as a storage for container images.
- **Repository** – an entity within a registry, in which the images are grouped. Typically, repository names are taken according to the OS family used on the images that will be stored in this or that repository (e.g., CentOS).
- **Tag** – images in repositories are referenced by tags, which are typically given according to image OS version (e.g., 6, 7, etc.)**.**

To start the Docker Service, use the **or2-manage-service (or2ms)** command with the following parameters:

```
or2-manage-service -p project -r region -s docker –c cluster-name
--activate
```

The command runs a Maestro Stack that creates and configures a Docker Master VM and stats a new cluster.

If you need a new node to be added to your Docker cluster, just repeat the **or2-manage-service** command.

By default, Docker nodes are MEDIUM-shaped VM with Ubuntu 14.04 operating system. If needed, you can use the **--shape** parameter with the **or2-manage-service** command to run a node with a non-default shape.

To stop the service, use **--deactivate** and -**i instance_id** to deactivate nodes one by one, with the Docker Master being deactivated the last.

3.1.1 Working with Containers

The node VM plays the role of a base for containers that use the node resources. To run, terminate, stop and start the containers, the **or2-docker-container (or2dc)** command is used.

- To run a new container, call the **or2dc** command with the following parameters:

  ```
  or2-docker-container -p project -r region -a run -i image_id -cn
  cluster_name
  ```

  Here, the **-a** parameter specifies the action to be performed (run), the **-i** parameter specifies the container image to be used, and the **-cn** parameter refers to the cluster created during the service activation.

  Docker manager will automatically select the node with the lowest load and run the container there.

  To see the list of images available in the specified cluster, run the **or2di** command with the **-a describe** flag:

  ```
  or2-docker-image -p project -r region -a describe –cn cluster_name
  ```

While running a new container, you can specify an entry point command by setting it in the **--cmd** parameter:

```
or2-docker-container -p project -r region -a run -i image_id -cn
cluster_name --cmd "/bin/bash"
```

- To stop or start a container, use the **or2dc** command with the following parameters:

```
or2-docker-container -p project -r region -a stop[or start]
-c container_name -cn cluster_name
```

The **stop** command shuts down the container and releases some of the resources occupied by it. The **start** command re-launches the stopped container.

- To remove a container, call the **or2dc** command with the following parameters:

```
or2-docker-container -p project -r region -a terminate
-c container_name -cn cluster_name
```

Here, the **-a** parameter specifies the action to be performed (terminate) and **-c** specifies the container name or ID.

3.1.2 Working with Docker Images via Docker Registry

Container creation requires an image describing the software which has to be installed in the container for the application deployment. Images are stored in public or private registries from where they can be shared between clusters.

In EPAM Cloud, Docker service is provided together with the **Docker Registry** facilities. A **Registry** is created on a separate VM and is used as a storage for images. Images in a registry are organized as a catalog: they are grouped in **repositories** and **tagged**.

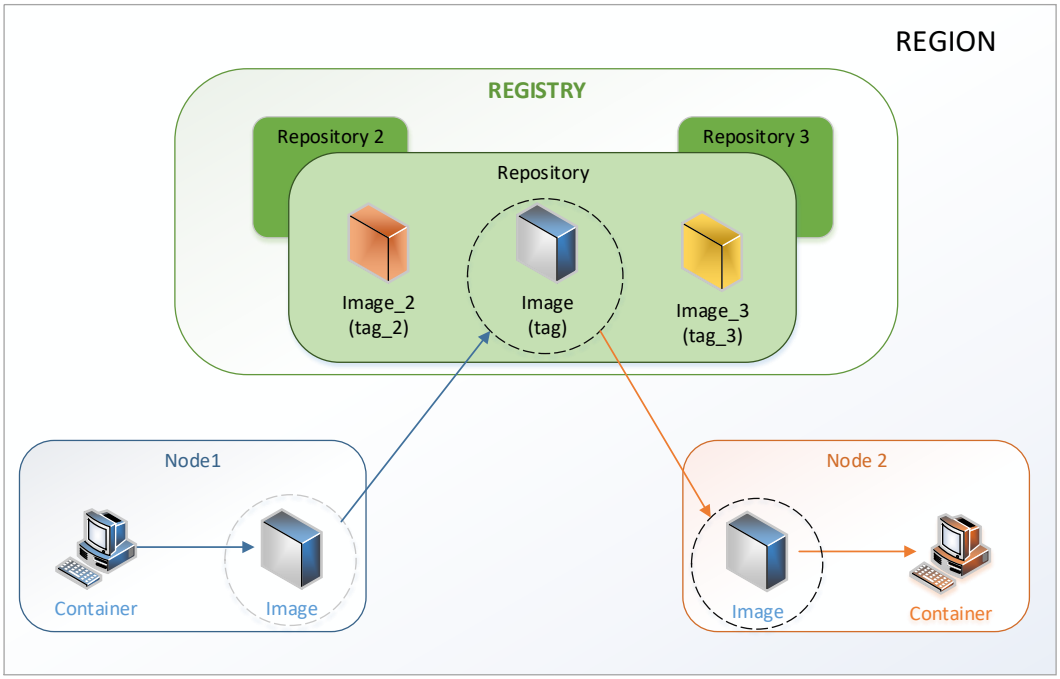

Figure 1 - Docker Registry flow

We typically recommend to name repositories after the OS family (e.g. CentOS), and assign tags to images according to the OS version (6,7, etc.).

Docker service and registry manipulations are performed with a set of the following Maestro CLI commands:

The typical Docker images manipulation flow can be described in the following steps:

1. Create a new Docker Registry:

```
or2ms -p project -r region -a –s docker-registry
```

On the command call, Orchestrator launches a **SMALL** VM based on **Ubuntu12.04_64-bit** image. The command response will return the ID of the stack, used to create the new registry:

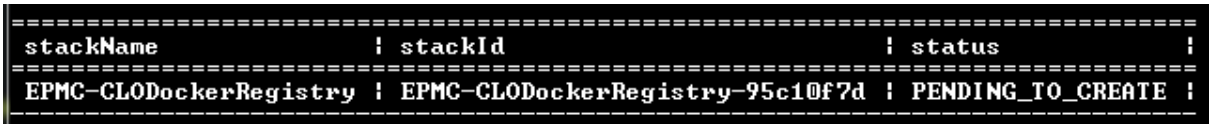

Figure 2 - Docker registry creation

2. Find the DNS of the Registry VM, using the **or2-describe-sevices (or2dser)** command:

```
or2dser -p project -r region
```

The command output will give the list of active project services, where you can find the details:

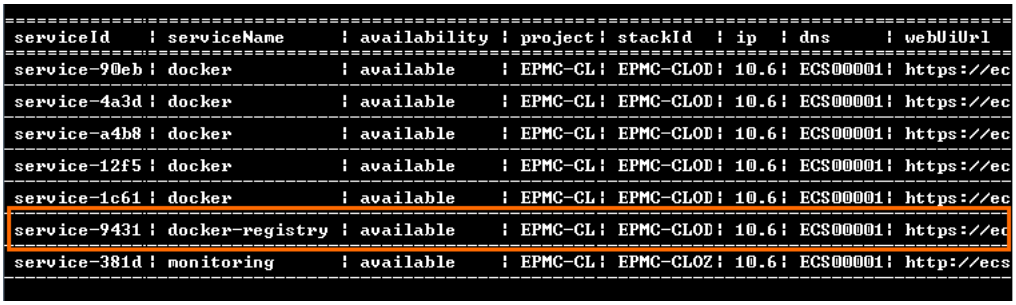

Figure 3 - Docker registry in the list of services

3. Create a new image from an existing container, using **or2-docker-image (or2di)** command with **-a commit** property:

```
or2di -p project -r region -a commit -c container_id -n image_name
-t image_tag –cn cluster_name
```

Where:

- **-a –** the action to be taken
- **-c** – the ID of the container that will be used to create an image
- **-i** – the ID of the image to be created
- **-t** – the tag of the image to be created
- **-cn** – the name of the Docker cluster to be used

When the command is executed, you will find the following response:

```
+-----------------------------------+
| id           | name     | tag     |
+-----------------------------------+
| 7ac3c78e4f7e | my_image | my_tag  |
+-----------------------------------+
```

Figure 4 - Docker image creation

4. Push the new image to an existing Registry, using **or2-docker-image (or2di)** command with **-a push** property:

```
or2di -p project -r region -a push -R target_repo_name
-i image_name -t image_tag -dr dockerRegistryInstanceID -cn
cluster_name
```

You can check whether the image was uploaded to the registry by calling the **or2-describe-registry-image (or2dri)** command:

```
or2dri -p project -r region -dr registry_instance_id
```

Please note, that on the registry, the image name is treated as a repository name:

```
>or2di -r EPAM-BY1 -p DEMOPRO -a push -i my_ubuntu -t 12.04 -dr ECS002.epam.com
Response: Push image my_ubuntu:12.04 on docker registry ECS002.epam.com is in progress.
See "or2dri -a describe -i ECS002.epam.com" for details.

> or2dri -a describe -i ECS002.epam.com -r EPAM-BY1 -p DEMOPRO

+-----------------------------------------------------------+
| host            | id           | repository | tag    |
+-----------------------------------------------------------+
| ECS002.epam.com | fa78bb972beb | my_ubuntu  | 12.04  |
+-----------------------------------------------------------+
```

Figure 5 - Image pushed to Docker registry

5. Pull the image from the Registry to all existing nodes, using **or2-docker-image (or2di)** command with **-a pull** property:

```
or2di -p project -r region -a pull -cn cluster_name
-i image_name -t image_tag -dr dockerRegistryInstanceID
```

After this, you can check the images on other nodes and find the imported one there. Please note that the image name includes the hostname of the registry where it is taken from:

```
> or2dri -a describe -i ECS003.epam.com -r EPAM-BY1 -p DEMOPRO
+-------------------------------------------------------------------------------+
| host            | id           | name                                | tag    |
+-------------------------------------------------------------------------------+
| ecs003.epam.com | fa78bb972beb | ecs002.epam.com:5000/my_ubuntu      | 12.04  |
| ecs003.epam.com | b0bc9f7e26f0 | oracle_sshkey_epc                   | 6      |
| ecs003.epam.com | f34ebf13eac2 | ubuntu_sshkey_epc                   | 12.04  |
| ecs003.epam.com | 77e72ad6fec0 | centos_sshkey_epc                   | 6      |
| ecs003.epam.com | b20f030c9cd8 | debian_sshkey_epc                   | 7      |
| ecs003.epam.com | 9eefca874fc6 | ecs000010484.epam.com:5000/oracle   | 6      |
| ecs003.epam.com | 0b310e6bf058 | ecs000010484.epam.com:5000/ubuntu   | 12.04  |
| ecs003.epam.com | 61f7f4f722fb | ecs000010484.epam.com:5000/debian   | 7      |
| ecs003.epam.com | 539c0211cd76 | ecs000010484.epam.com:5000/centos   | 6      |
+-------------------------------------------------------------------------------+
```

Figure 6 - Image pulled from Docker registry

After this, the new image can be used to run containers in the Docker cluster.

You can also pull images from **public** Docker Registry (https://index.docker.io). To do this, run the **or2di -a pull** command without the **-dr** parameter.

6. To delete a repository with all included images (tags) from a registry, run the **or2dri** command with the the **-a delete** parameter:

```
or2dri -p project -r region -a delete –cn cluster_name
-dr dockerRegistryInstanceId -R repository_name
```

You can delete only the registry with all included tags. It is impossible to remove a single tag (image)

3.1.3 Working with Docker Volumes

When a container is used, changes to its data are not stored and may be lost if the container is deleted. Running a container from an image will not include any modified data.

In order to save the data and share it between the Docker containers, Docker Swarm volumes are used. A volume is a directory within a container mapped to a directory on a host. The data is stored in the volume and can be used later.

Volume management is performed with the **or2-docker-volume (or2dv)** command.

To create a Docker volume, run the **or2dv** command with the following parameters:

```
or2-docker-volume -p project -r region -a create –cn cluster_name
–v volume
```

```
Response:


+-------------------------------------------------------------------------------------------------+
| host                    | name        | driver | mountPoint                                   |
+=================================================================================================+
| evuakha294d11.novalocal | testvolume2 | local  | /var/lib/docker/volumes/testvolume2/_data |
+-------------------------------------------------------------------------------------------------+
```

Figure 7 - Docker volume creation

To view the list of all volumes available on the Docker Swarm for the specified project and region, run the **or2dv** command with the following parameters:

```
or2-docker-volume -p project -r region -a describe –cn cluster_name
```

```
Response:


+-------------------------------------------------------------------------------------------+
| host                    | name        | driver | mountPoint                             |
+-------------------------------------------------------------------------------------------+
| evuakha294d11.novalocal | testvolume  | local  | /var/lib/docker/volumes/testvolume/_data  |
+-------------------------------------------------------------------------------------------+
| evuakha294d11.novalocal | testvolume2 | local  | /var/lib/docker/volumes/testvolume2/_data |
+-------------------------------------------------------------------------------------------+
```

Figure 8 - List of Docker volumes

To delete the specified Docker volume, run the **or2dv** command with the following parameters:

```
or2-docker-volume -p project -r region -a delete –cn cluster_name
–v volume
```

```
D:\cli\maestro-local\bin> or2dv -cn cluster1 -a delete -v testvolume2
Adding default parameter: --project
Adding default parameter: --region
Are you sure you want to delete this volume? Type "yes" for confirmation: yes
Date: 2016-05-13T14:43:30+00:00
Response:


+-------------------------------------------------------------------+
| name        | driver | mountPoint                                 |
+-------------------------------------------------------------------+
| testvolume2 | local  | /var/lib/docker/volumes/testvolume2/_data |
+-------------------------------------------------------------------+
```

Figure 9 - Docker volume deletion

Docker Swarm volumes are available only in OpenStack-based regions.

Docker volumes are mapped to host directories by means of binding performed during the **or2dc** command execution. To bind a volume, run the **or2dc** command with the **-v (--volume)** parameter.

```
D:\cli\maestro-local\bin> or2dc -cn cluster1 -a run -i nginx:1.7 -v testvolume:/etc/nginx
Adding default parameter: --project
Adding default parameter: --region
Date: 2016-05-13T14:47:31+00:00
Response:


+-------------------------------------------------------------------+
| host                    | id           | name                 |
+-------------------------------------------------------------------+
| evuakha294d11.novalocal | 045d16572764fc65 | condescending_snyder |
+-------------------------------------------------------------------+
```

Figure 10 - Running Docker container with a volume

3.1.4 Docker Info

The **or2-describe-docker (or2dd)** command allows to get the list of the existing Docker elements and their details.

Using this command with different parameters, you can get the following information:

- General overview of the Docker service resources with their details and roles:

```
or2dd -p project -r region -cn cluster_name
```

```
D:\maestro-cli\bin>or2dd -p EPM-CSUP -r DEMOREG
Response:

+-------------+-------------------------+-------------+---------+-----+--------+-------------------------+
| instanceId  | dnsName                 | privateIp   | state   | cpu | memory | description             |
+-------------+-------------------------+-------------+---------+-----+--------+-------------------------+
| i-25a9ac1a  | ECS000010B25            | 10.6.133.46 | running | 2   | 3840   | Project Docker Swarm Agent  |
| i-6f26b49d  | ECS000010B23.epam.com   | 10.6.133.68 | running | 2   | 3840   | Project Docker Swarm Master |
+-------------+-------------------------+-------------+---------+-----+--------+-------------------------+
```

Figure 11 - List of Docker resources

- The detailed information on a Docker cluster:

```
or2dd -p project -r region -t cluster -cn cluster_name
```

```
D:\maestro-cli\bin>or2dd -p EPM-CSUP -t cluster
Response:

+-----------------------------------------+----------+------------+-----------------+
| filters                                 | strategy | nodesCount | containersCount |
+-----------------------------------------+----------+------------+-----------------+
| affinity, health, constraint, port, dependency | spread   | 2          | 4               |
+-----------------------------------------+----------+------------+-----------------+
```

Figure 12 - Docker cluster data

3.1.5 REST API

In EPAM Cloud, Docker manipulations are performed via Maestro CLI.

However, there is a REST API you can use, if needed. The connection details can be found in **or2dser** command response.

Access to the REST API is established via SSL and requires the CA certificate, the host certificate signed by the same CA certificate and the host private key:

```
/etc/docker/ca.crt /etc/docker/host.crt /etc/docker/host.key
```

For Docker Swarm, use the following request:

```
curl --cacert /etc/docker/ca.crt --cert /etc/docker/host.crt --key
/etc/docker/host.key https://{INSTANCE_PUBLIC_IP}:4000/info
```

For Docker host, use the following request:

```
curl --cacert /etc/docker/ca.crt --cert /etc/docker/host.crt --key
/etc/docker/host.key https://{INSTANCE_PUBLIC_IP}:2376/info
```

3.1.6 Pricing

The service usage price is defined by the price of the Docker node VM.

The default parameters of a Docker node VM are:

- Shape: MEDIUM
- Image: CoreOS_1632_64-bit

Therefore, the approximate monthly cost of a Docker Server usage in case of 100% and 24/7 load is about $58.59 in EPAM-BY1 region (as to 11/09/2015). The price can vary depending on the region and the shape you select for the Docker node VM. To get more detailed estimations, please, use our Cost Estimator tool.

3.2 KUBERNETES SERVICE

If you use Docker as your containerization platform, Kubernetes may be one of your clustering options. Kubernetes groups containers to be managed as a single application into 'pods' which will be running on clusters of virtual machines.

Within a cluster, one virtual machine functions as a master node consisting of a stateless API server, scheduler and controller manager. Additionally, the master node includes an etcd key-value store for data storing in the cluster. The master node manages the workload and provides communication within the cluster through the API server. Other virtual machines are worker nodes subordinate to the master.

The basic concepts used in Kubernetes are as follows:

- **Master** – the main VM from which the Kubernetes environment is managed
- **Node** – VM on which containers are deployed and running
- **Cluster** – a set of virtual or physical machines on which applications are running
- **Pod** – one or several containers to be run as a single application and the related container options
- **Service** – the functionality within the Kubernetes environment balancing the load between the pods and providing DNS names
- **Replication Controller** – the entity setting the number of pods to be running
- **Networking** – the service allowing to configure IP address ranges and the related settings
- **Namespace** – a virtual cluster within a physical cluster

3.2.1 Service Activation

Kubernetes as a Service can only be activated in OpenStack private regions. For a cluster proper configuration, you need to activate Kubernetes Master and at least one worker (UI Dashboard, Proxy, Addons for DNS).

To start the Kubernetes Service, use the **or2-manage-service (or2ms)** command with the following parameters:

```
or2-manage-service -p project -r region -s kubernetes –c cluster-name
-k key_name --activate
```

To start the service as a Kubernetes master, run the command with the **-s** (**--service**) parameter containing '**kubernetes**' or '**kubernetes-master**'. To start the service as a Kubernetes worker, run the command with the **-s** (**--service**) parameter containing '**kubernetes-worker**'.

```
or2-manage-service -p project -r region -s kubernetes-worker –c
cluster_name -k key_name --activate
```

You can activate a worker node only if a master node has already been activated in the same cluster.

If you need a new node to be added to your Kubernetes cluster, just repeat the **or2-manage-service** command.

By default, Kubernetes nodes are **LARGE**-shaped VMs with **CoreOS_1632_64-bit** operating system.

To stop the service, use **--deactivate** and **-i instance_id** parameters to deactivate nodes one by one, with the Kubernetes Master being deactivated the last.

You can start more than one Kubernetes cluster for the same project and region.

3.2.2 Kubernetes Info

To view the list of the existing Kubernetes nodes and their parameters, run the **or2-describe-services** (**or2dser**) command specifying your project and region:

```
or2dser -p project -r region -s kubernetes
```

serviceId	serviceName	instanceSearchId	availability	stackId	ip
service-********	kubernetes-master	evuakha******	available	EPM-CIT2KubernetesMaster-********	10.6.*******
service-********	kubernetes-master	evuakha******	available	EPM-CIT2KubernetesMaster-********	10.6.*******

dns	keyName	webUiUrl	user	password	clusterName
10.6.*******	*******	https://10.6.*******:*****/	kubernetes	**********	**********
10.6.*******	*******	https://10.6.*******:*****/	kubernetes	**********	**********

Figure 13 - List of Kubernetes nodes (shown in two lines for better visibility)

3.2.3 Kubernetes Pods Management

To view the list of pods within a particular Kubernetes cluster, use the **or2-kubernetes-pods (or2kp)** command:

```
or2kp -p project -r region -cn cluster_name
```

The command will return the list of all Kubernetes pods within the cluster specified in the command, together with their state and start date.

Response:

nodeName	name	phrase	startDate
10.6.***.***	nginx-controller-99ia8	Pending	2016-11-14T08:29:05Z
10.6.***.***	nginx-controller-wwa2b	Pending	2016-11-14T08:29:05Z
10.6.***.***	kube-apiserver-10.6.***.***	Running	2016-11-10T17:22:47Z
10.6.***.***	kube-controller-manager-10.6.***.***	Running	2016-11-10T17:22:47Z
10.6.***.***	kube-dashboard-under-nginx	Running	2016-11-10T17:23:40Z
10.6.***.***	kube-proxy-10.6.***.***	Running	2016-11-10T17:22:47Z
10.6.***.***	kube-scheduler-10.6.***.***	Running	2016-11-10T17:22:47Z

Figure 14 - List of pods in a Kubernetes cluster

3.2.4 Kubernetes Namespaces Management

To view the list of all namespaces (virtual clusters) within the particular physical cluster, use the **or2-kubernetes-namespaces (or2kns)** command:

```
or2kns -p project -r region –cn cluster_name
```

The response will contain the list of namespaces and their state.

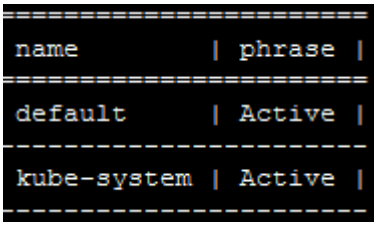

Figure 15 - List of namespaces in a Kubernetes cluster

3.2.5 Kubernetes Services Management

To view the list of services in the particular Kubernetes cluster, run the **or2-kubernetes-services (or2ks)** command:

```
or2ks -p project -r region –cn cluster_name
```

The command returns the list of Kubernetes services in the specified cluster and their types.

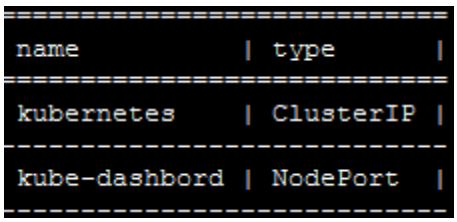

Figure 16 - List of services in a Kubernetes cluster

3.2.6 Kubernetes Replication Controllers Management

Replication controllers are used to run Kubernetes pods from the so-called pod templates, thus creating the specified number of identical pods. To view the list of replication controllers in the particular Kubernetes cluster, use the **or2-kubernetes-replication-controllers (or2krc)** command:

```
or2krc -p project -r region –cn cluster_name
```

The command returns the list of all replication controllers used in the cluster, the pod template used for replication, the target number of replicas, the number of currently existing replicas and the number of completed replicas.

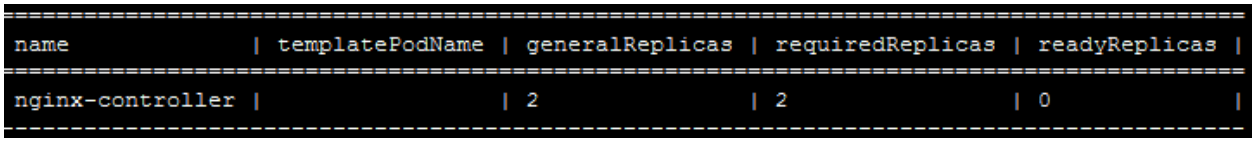

Figure 17 - List of replication controllers

3.2.7 Kubernetes Nodes Management

To view the list of nodes within the specified Kubernetes cluster, use the **or2-kubernetes-nodes (or2kn)** command:

```
or2kn -p project -r region –cn cluster_name
```

The command response contains the list of nodes, their state, number of pods in each node, the CPU count and the memory size.

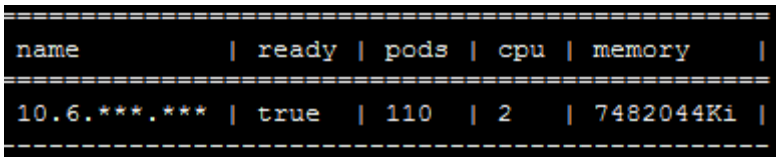

Figure 18 - List of Kubernetes nodes

3.2.8 Using the Service

When the Kubernetes service is activated, you can use it via the native command line interface, **kubectl**, which is automatically available on the master virtual machine on which Kubernetes is activated.

For other machines, download the kubectl executable and install it.

For more details on using **kubectl**, on the available commands, syntax and parameters, refer to the kubectl Overview page of the official Kubernetes website.

3.2.9 kubectl Configuration

Before using Kubernetes via the local kubectl, configure kubectl to work with the cluster using the following kubectl commands:

```
kubectl config set-cluster default-cluster --
server=https://MASTER_PUBLIC_IP --certificate-
authority=/etc/kubernetes/ssl/ca.pem
```

where MASTER_PUBLIC_IP is the public IP of the master node which you are accessing.

```
kubectl config set-credentials default-admin --certificate-
authority=/etc/kubernetes/ssl/ca.pem --client-
key=/etc/kubernetes/ssl/admin-key.pem --client-
certificate=/etc/kubernetes/ssl/admin.pem
```

```
kubectl config set-context default-system --cluster=default-cluster
--user=default-admin
```

```
kubectl config use-context default-system
```

The commands use the SSL credentials stored in the following files:

```
/etc/kubernetes/ssl/ca.pem

/etc/kubernetes/ssl/admin-key.pem

/etc/kubernetes/ssl/admin.pem
```

These files can be obtained from the master node.

To find the certificate Common Name, use the following command:

```
openssl x509 -noout -subject -in ~/.kube/admin.pem
```

3.2.10 RBAC Authorization

Kubernetes supports RBAC (Role-Based Access Control) authorization allowing dynamic permission policies configuration using the Kubernetes API. This authorization approach allows achieving higher security and differentiated access for users. Users can be added to groups with access to different functions or to different workspaces. Access can be revoked or modified for a single user without changing the certificate. For more details on RBAC authorization for Kubernetes, refer to the Kubernetes documentation.

For users who had already working Kubernetes clusters before the upgrade, we recommend performing the following operations to keep their systems working. These operations are intended to set the proper ClusterRoles and to configure the ClusterRoleBinding to continue having the admin access:

ClusterRole:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1alpha1
metadata:
  name: admin-access
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]
    nonResourceURLs: ["*"]
```

ClusterRoleBinding:

```
apiVersion: rbac.authorization.k8s.io/v1alpha1
kind: ClusterRoleBinding
metadata:
  name: admin-access
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
roleRef:
  apiVersion: rbac.authorization.k8s.io/v1alpha1
  kind: ClusterRole
  name: admin-access
```

3.2.11 Web UI

The Kubernetes service has a web UI which is automatically available as soon as the service is activated in the cluster. It is accessible via a URL over the https connection. The Web UI URL can be found from the Kubernetes node details returned by the **or2dser** command.

The web UI has most of the functionality supported by the CLI in an intuitive format.

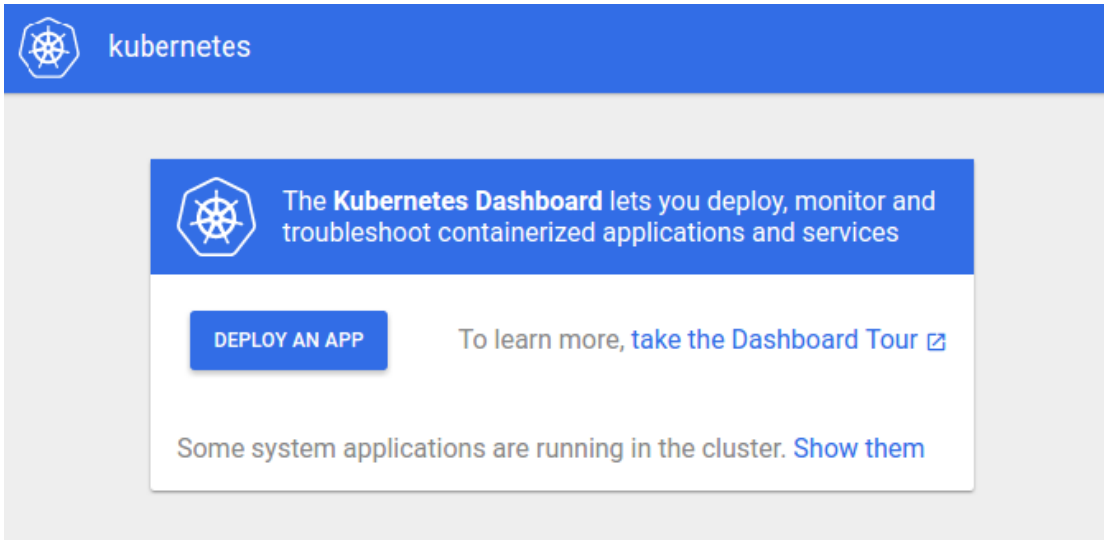

Figure 19 - Kubernetes Dashboard

Clicking 'Deploy an App' opens the application deployment wizard helping you to set up your application either by specifying its parameters manually or by uploading a YAML or JSON file containing the application configuration.

When the application is running, you can use the Kubernetes Dashboard to monitor its performance, debug errors and delete applications.

Kubernetes Dashboard supports basic authentication using login and password. With service activation, a login and password are generated to be used for accessing the Web UI. The login and password for each Kubernetes node can be retrieved by means of the **or2-describe-services (or2dser)** command.

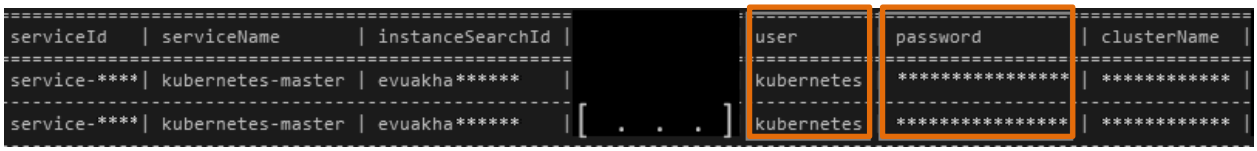

Figure 20 - Username and password in or2dser response

3.2.12 Container Images

During Docker container management Kubernetes refers to the Docker repository where Docker container images are stored. In the current implementation, Kubernetes supports image retrieval from the public Docker registry available at https://index.docker.io.

3.2.13 Troubleshooting

If the Kubernetes service is not performing as expected, you can try to identify the root cause of the issue and, in certain cases, debug it yourself. The first step is to determine whether the issue concerns your application or your cluster.

If the application is not performing correctly, run the following kubectl commands on your master VM to diagnose the application:

Command	Description
# kubectl get pods	Lists the pods included in your application with their statuses
# kubectl get services	Lists the services running in your Kubernetes environment
# kubectl get replicationControllers	Lists the replication controllers and the containers they have started
# kubectl describe pods $ {pod_name}	Displays the status of a particular pod

If pods or replication controllers are not performing properly, the cause may be insufficient resources. This can be resolved by adding more nodes to the cluster or terminating the pods which are no longer needed.

If the issue occurred in the Kubernetes services, check whether the service exists and if it is correctly configured.

Otherwise, contact the Level 3 Support Team.

If the issue is within the cluster, run the following kubectl commands on your master VM to diagnose the nodes:

Command	Description
# kubectl get nodes	Lists the nodes in the cluster with their statuses

Contact the Level 3 Support Team for assistance with node issues.

3.2.14 Pricing

The Kubernetes service is available in the OpenStack private regions and is billed according to the VM billing in that region. The default parameters of a Kubernetes VM are:

- Image: CoreOs_1632_64-bit
- Shape: LARGE

Therefore, the approximate monthly cost in case of 100% and 24/7 load is about $61.04 in EPAM-BY2 region (as of April 4, 2018). To get more detailed estimations, please, use our Cost Estimator tool.

TABLE OF FIGURES

Figure 1 - Docker Registry flow 7
Figure 2 - Docker registry creation 8
Figure 3 - Docker registry in the list of services 8
Figure 4 - Docker image creation 9
Figure 5 - Image pushed to Docker registry 9
Figure 6 - Image pulled from Docker registry 9
Figure 7 - Docker volume creation 10
Figure 8 - List of Docker volumes 11
Figure 9 - Docker volume deletion 11
Figure 10 - Running Docker container with a volume 11
Figure 11 - List of Docker resources 12
Figure 12 - Docker cluster data 12
Figure 13 - List of Kubernetes nodes (shown in two lines for better visibility) 15
Figure 33 - List of pods in a Kubernetes cluster 15
Figure 34 - List of namespaces in a Kubernetes cluster 16
Figure 35 - List of services in a Kubernetes cluster 16
Figure 36 - List of replication controllers 16
Figure 37 - List of Kubernetes nodes 17
Figure 14 - Kubernetes Dashboard 19
Figure 15 - Username and password in or2dser response 19

VERSION HISTORY

Version	Date	Summary
1.3	April 4, 2018	– Updated information about Kubernetes
1.2	December 16, 2016	– Description of Kubernetes commands added – or2dr command removed – Classification changed from Confidential to Public, approved by Dzmitry Pliushch
1.0	June 1, 2016	First published