Datawatch Corporation makes no representation or warranties with respect to the contents of this manual or the associated software and especially disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Datawatch Corporation reserves the right to revise this publication and make changes from time to time to its contents without obligation to notify anyone of such revisions or changes.

Datawatch Monarch Swarm Enterprise Server software is offered and is to be used in accordance with a SOFTWARE LICENSE AND MAINTENANCE AGREEMENT. This agreement stipulates that this software be used only in the computer system designated in that agreement. The agreement further stipulates that the customer shall not copy or alter, or permit others to copy or alter, the software or related materials in whole or in part, in any media for any purpose, except to make an archive (back-up) copy or to make a copy as an essential step in the use of the software with the customer's computer.

Datawatch Corporation hereby grants the buyer the right to reprint this documentation for internal uses only. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, for any other purposes, without the prior written permission of Datawatch Corporation.

**Datawatch Monarch Swarm Enterprise Server v2.2 Installation Guide**

Copyright © 2018 by Datawatch Corporation

All rights reserved. Printed in the U.S.A.

Unpublished - Rights reserved under the copyright law of the United States.

Datawatch Monarch Swarm Enterprise Server is a trademark of Datawatch Corporation. Other products mentioned herein may be trademarks or registered trademarks of their respective owners in the United States or other countries.

For U.S. Government End Users, the software is a "Commercial Item(s),” as that term is defined at 48 C.F.R. Section 2.101, consisting of "Commercial Computer Software” and “Commercial Computer Software Documentation,” as such terms are used in 48 C.F.R. Section 12.212 or 48 C.F.R. Section 227.7202, as applicable. Consistent with 48 C.F.R. Section 12.212 or 48 C.F.R. Sections 227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the Datawatch Software License and Maintenance Agreement.
Table of Contents

Monarch Swarm Enterprise Server Installation Guide ........................................ 1

Components ............................................................................................................. 1
  Monarch Swarm Enterprise Server Cluster components .......................................... 2
  Deployment Diagram .............................................................................................. 4
Multitenancy Approach .............................................................................................. 5
Horizontal Scalability Approach ............................................................................... 6
  Scalability Per Component ..................................................................................... 6
Benefits of Kubernetes ............................................................................................... 8

Installation .................................................................................................................. 9

Requirements ............................................................................................................ 9
  Deployment ............................................................................................................... 9
Configuring Ansible ..................................................................................................... 10
Kubernetes Installers .................................................................................................. 13
  AWS Installation ..................................................................................................... 13
  Standalone Installation ........................................................................................... 15
  On-Premise Installation .......................................................................................... 16
Swarm Pre-Installation Modules ............................................................................... 18
  Kubernetes Modules ................................................................................................. 18
  The Kubernetes Dashboard ....................................................................................... 21
  Logging ..................................................................................................................... 22
  Monitoring ................................................................................................................ 24
Persistence Storage Configuration ............................................................................... 25
  Configuring iSCSI ..................................................................................................... 25
  Configuring Ceph Remote ......................................................................................... 25
Monarch Swarm Enterprise Server Installation ....................................................... 26
  Monarch Swarm Installation Steps ........................................................................... 26
  Swarm Cleanup In Case of Installation Errors ........................................................... 28
  Configuring Single Sign-On (SSO) .......................................................................... 29
  S3 File Library Configuration ................................................................................. 31
  Application properties ............................................................................................. 32
  Creating New Tenants ............................................................................................. 45

Server Maintenance ................................................................................................... 46

Utilities ....................................................................................................................... 46
  Maintaining Monarch Swarm Enterprise Server .................................................... 47
  Upgrading Monarch Swarm Enterprise Server ....................................................... 48
Monarch Swarm Enterprise Server Installation Guide

This installation guide describes how to install and deploy the Monarch Swarm Enterprise Server application.

Components

Monarch Swarm Enterprise Server includes several components:

- The Monarch Swarm Enterprise Server application
- ELK Stack for logging
- Kubernetes Dashboard for system administration

**Ansible 2.5** is used to install and update server and Monarch Swarm Enterprise application. Monarch Swarm Enterprise server is built on a Kubernetes cluster. The **Kubeadm tool** is used to create servers. You can use the command line tool **kubectl** to deploy and manage applications in Kubernetes.

Using **kubectl**, you can:

- Inspect cluster resources
- Create, delete and update components
- Look at your new cluster
- Bring up example apps

These components are packaged in [Helm Package Manager](https://helm.sh) for Kubernetes with custom configurations.
MONARCH SWARM ENTERPRISE SERVER CLUSTER COMPONENTS

The following cluster components are included in Monarch Swarm Enterprise Server

- ingress – Nginx controller for the Monarch Swarm Enterprise server application
  - Ingress for Monarch Swarm clusters
  - The only component that is visible to the outer world
  - Used for TLS termination
- core http-api – Spring Boot 2 application
  - Serves all Monarch Swarm core functionalities (HTTP and WebSocket endpoints)
  - Swarm library (workspaces, data sources, connections, change lists, folders, etc.)
  - User management (users, roles, groups)
  - Process management (processes, schedules, etc.)
  - Data readers (CSV, excel, JDBC, etc.)
  - Data writers (CSV, excel, JDBC, etc.)
  - Report trapping
  - Data fetching and querying (design mode, batch mode, statistics)
- core api postgres – PostgreSQL server
  - Used to store core api metadata objects
- data-engine http-api – Spring Boot 2 application
- data-engine api postgres – PostgreSQL server
  - Used to store data-engine api metadata objects
- data-engine worker – Spring Boot 2 application
  - Data Engine for design mode
- data-engine postgres – PostgreSQL server with PostgreSQL PL/Java
  - Data engine backend used to store data sources data and query workspaces relational tree for design mode
- data-engine batch – Spring Boot 2 application and PostgreSQL server with PostgreSQL PL/Java
  - Data Engine for batch requests
  - Data engine backend used to store data sources data and query workspaces relational tree for export
machine-learning http-api – **Spring Boot 2** application
- Machine learning recommendations and social functionality (likes, follows, etc.) of the Monarch Swarm cluster

Distributed cache – **Redis** server
- Used in the core http-api distributed mode as a hibernate l2 cache (metadata storage), spring cache (data source preview)
- Used in data-engine http-api distributed mode for distributed locks, maps, counters (data-engine).

Spark cluster – **Apache Spark** analytics engine
- Spark driver for machine-learning recommendations
- Spark drivers for data-engine batch mode

Cassandra cluster – **Apache Cassandra** database
- Used as backend storage to import legacy workspaces from Monarch Complete
- Used as backend storage for published data sources (internal library)
- Used as backend storage for machine-learning http-api
- Used as backend storage for machine-learning recommendations

Logging server – **ELK Stack** with **Fluentd** data collector
- Used as a persistent search and analytics engine
- Used as a log viewer for visualizing logs with charts and graphs
- Used to unify data collections

rabbitmq server – **RabbitMQ** message broker
- Used for asynchronous request for batch and design mode requests
Multitenancy Approach

Our current multitenancy approach is highly based on virtualization with the aim of reducing costs for supporting both single-tenant and multi-tenant versions of the Monarch Swarm Enterprise Server application. Such an approach gives us relatively zero-development to support this functionality (with the help of Kubernetes namespaces).

The following approach is taken:

- Each tenant has its own virtual cluster (i.e., all of the components on the deployment diagram)
- All of the tenants' clusters are isolated between each other (i.e., each tenant has its own database, possibly its own codebase to allow some customization between tenants, etc.)
- Each tenant has own resource quotas (CPU, memory, storage) in the cluster
- Each tenant can scale components in cluster independently from other tenants (which allows more flexibility in setting up optimal resources for a particular tenant workload)
- The tenant administrator has full access to manage tenants (create, delete, scale, define resource quotas, etc.)

Additional Information

- Namespaces in kubernetes
- Resource quotas in the namespace
Horizontal Scalability Approach

Each component in the cluster can be scaled with known limitations.

Two types of scalability are supported:

- Manual scaling: driven by cluster/tenant administrator
- Automatic scaling: based on resource consumption (e.g., CPU) or specific metrics (e.g., request rate)

The scalability approach differs for stateful and stateless components:

- Stateful components (databases): Distributed cache, data-engine batch, cassandra cluster, rabbitmq-ha

Spark is a unique component that already has some clustered solutions with the scalability approach:

- [https://github.com/apache-spark-on-k8s/spark](https://github.com/apache-spark-on-k8s/spark)

**SCALABILITY PER COMPONENT**

- **ingress** — Nginx is scalable by design
- **core http-api** — This component is stateless by design, so it can be scaled with some limitations
  - core api postgres connections limit
  - core api postgres throughput
- **core api postgres** — metadata storage for core http-api; this component is not scalable
- **data-engine http-api** — this component is stateless by design so can be scaled with some limitations:
  - data-engine postgres connections limit
  - data-engine postgres throughput
- **data-engine api postgres** — metadata storage for data-engine http-api, this component is not scalable
- **data-engine worker** — data-engine for design-mode requests is stateless and scalable by design
- **data-engine postgres** — this component is not scalable by design
- **data-engine batch** — data-engine for batch process can be scaled
- machine-learning http-api — is not scalable by design
- spark cluster — Spark is scalable by design
- cassandra cluster — Cassandra is scalable by design
- rabbitmq-ha - message broker is scalable by design
- distributed cache — Redis is scalable by design

Additional Information
- https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
- https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling
- https://redis.io/topics/cluster-spec
- https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/
- https://kubernetes.io/docs/tutorials/stateful-application/cassandra/

Kubernetes deployments/statefulset with scaling support:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Support Scaling</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>core-api</td>
<td>stateless</td>
<td>true</td>
<td>–</td>
</tr>
<tr>
<td>core-api-postgres</td>
<td>stateless</td>
<td>false</td>
<td>core-api database</td>
</tr>
<tr>
<td>data-engine-api</td>
<td>stateless</td>
<td>true</td>
<td>Should be scaled on each node in the cluster</td>
</tr>
<tr>
<td>data-engine-api-postgres</td>
<td>stateless</td>
<td>false</td>
<td>data-engine-api database</td>
</tr>
<tr>
<td>data-engine-worker</td>
<td>stateless</td>
<td>true</td>
<td>Should be scaled on each node in the cluster</td>
</tr>
<tr>
<td>data-engine-batch</td>
<td>stateful</td>
<td>true</td>
<td>Should be scaled on each node in the cluster</td>
</tr>
<tr>
<td>rabbitmq-ha</td>
<td>stateful</td>
<td>true</td>
<td>Should be scaled on each node in the cluster</td>
</tr>
<tr>
<td>machine-learning</td>
<td>stateless</td>
<td>false</td>
<td>–</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Support Scaling</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>-----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>depostgres</td>
<td>stateless</td>
<td>false</td>
<td>Data engine for design mode, work with data-engine-worker.</td>
</tr>
<tr>
<td>cassandra</td>
<td>stateful</td>
<td>false</td>
<td>Support replication</td>
</tr>
<tr>
<td>redis</td>
<td>stateful</td>
<td>false</td>
<td>Support master-slave replication with persistence on file system</td>
</tr>
</tbody>
</table>

**Benefits of Kubernetes**

*Kubernetes (k8s)* is used as a cluster container orchestrator for:

- Automatic deployment
- Horizontal scaling
- Multitenancy (*namespace*-based)
- Storage orchestration
- Release management (*helm*-based)
- Self-healing
- Service discovery and load balancing
- Secret and configuration management
Installation

Requirements

To setup Monarch Swarm Enterprise Server, you need:

- Kubernetes cluster version 1.10.11. If you don’t have a Kubernetes cluster, use Kubernetes installers to set up the application. The minimum requirements for Monarch Swarm Enterprise Server are the following: 24 CPU and 80 Gb RAM reserved in the worker nodes of the Kubernetes cluster.

- Configure Ansible, Helm manager, and kubectl on the control machine (this machine should have full access to the kubernetes cluster). The Helm manager and kubectl can be set up with other Swarm pre-installation modules.

- Configure Static Persistence Volumes or Kubernetes Dynamic Volume Provisioning. Monarch Swarm Enterprise Server uses a minimum of 6 volumes: 4 for PostgreSQL data, 1 for Cassandra data, and 1 for the NFS server. All volumes should have ReadWriteOne access.

- A DNS provider to configure the Monarch Swarm Enterprise Server domain name. For AWS, it is Route53; for on-prem configuration, it is any other DNS provider. For on-prem, Kubernetes master/worker nodes should be resolved in DNS. For aws, Route 53 will be configured after Monarch Swarm Enterprise Server deployment.

DEPLOYMENT

The deployment process contains 4 main steps. All steps should be run sequentially:

- Configure Ansible. This step is required.

- Setup Kubernetes. This step is optional. If you already have a configured cluster, skip this step.

- Setup pre-installation modules. This step is optional. If you already have all pre-installation modules, skip this step.

- Setup Monarch Swarm Enterprise Server. This step is required.
Configuring Ansible

Ansible and Helm manager are used to set up the Kubernetes server, additional modules, and Monarch Swarm Enterprise Server.

These configurations should be set in the controller machine.

Set up and configure Ansible:

- Make sure the `sudo` has been configured. To check:
  
  ```
  $ sudo whoami
  ```

  and enter the `sudo password`.

  [How to Create a Sudo User on Ubuntu](#)

- **Ansible 2.5.2 is required.** Use `pip` to install Ansible:

  ```
  $ sudo curl https://bootstrap.pypa.io/get-pip.py | sudo python - 'pip==8.1.2'
  $ sudo pip install ansible==2.5.2
  ```

- For remote installation over SSH. Configure master and worker SSH access by certificate (preferable) or username/password with root access.

  ```
  $ ssh-keygen -t rsa -b 2048 -q -N "" -f ./swarm_server
  $ ssh-copy-id -i swarm_server root@<swarm_master_server_url>
  # run this for each worker
  $ ssh-copy-id -i swarm_server root@<swarm_worker_server_url>
  ```

  [How to Set Up SSH Keys on Ubuntu 16.04](#)

**AWS Ansible .conf/inventory Configuration**

1. Set server connection:

   ```
   [master]
   # Access by SSH with certificate
   master1 ansible_connection=local
   ```

2. Configure sudo access for control machine:

   ```
   [master:vars]
   ansible_become_user=<become_user>
   ansible_become_pass=<become_password>
   ansible_become_method=sudo
   ```
On-Prem Ansible ./conf/inventory Configuration

1. Set server connection. This connection type is preferred:

   [master]
   # Access by SSH with certificate
   master1 ansible_connection=ssh ansible_host=<ssh server host>

2. Configure workers; you can set any number of workers:

   [worker]
   #
   # Access by SSH with certificate
   node1 ansible_connection=ssh ansible_host=<worker1 host>
   node2 ansible_connection=ssh ansible_host=<worker2 host>

3. Set key pair for the server:

   [master:vars]
   # Private key for SSH type connection
   ansible_ssh_private_key_file=./conf/linuxserver.pem
   ansible_become_pass=<sudo password> # set sudo password

4. Set default user:

   [all:vars]
   #
   # Default user on the server. The user must exist inside the server
   ansible_user=ubuntu

Standalone Linux Ansible ./conf/inventory Configuration

1. Remote installation. For access by SSH with certificate:

   - Set server connection:

     [master]
     # Access by SSH with certificate
     master1 ansible_connection=ssh ansible_host=<ssh server host>

   - Set key pair for the server:

     [master:vars]
     # Private key for SSH type connection
     ansible_ssh_private_key_file=./conf/standalonelinuxserver.pem
     ansible_become_pass=<sudo password> # set sudo password

2. Remote installation. For access by SSH with username and password:

   [master]
   # Access by SSH with username/password
master1 ansible_connection=ssh ansible_host=<server host>
ansible_port=22 ansible_user=<user name> ansible_ssh_pass=<user password> ansible_become_pass=<sudo password>

3. Local installation. For installing on the local machine:

[master]
master1 ansible_connection=local ansible_become_pass=<sudo password>

4. For all configurations, ansible_user should be defined:

[master:vars]
# Default user on the server. The user must exist inside the server
ansible_user=ubuntu

NOTE: For more information on how to configure the Ansible inventory file, read the documentation.

To verify Ansible config, run these commands from Monarch Swarm Enterprise Server root installation folder:

1. Check user access:

   $ ansible -i ./conf/inventory master -a whoami -vvv

   This script should return your configured user. If not, check inventory connection and that you have configured SSH connection for Remote installation.

2. Check sudo access:

   $ ansible -i ./conf/inventory master --become -a whoami -vvv

   This script should return root user: "root". If not, check that you correctly configure sudo access and ansible_become_pass in the inventory file.

Additional Information

- Ansible Installation Guide
Kubernetes Installers

AWS INSTALLATION

These instructions will setup and configure Kubernetes cluster on AWS. Only Linux installation is supported.

Requirements
You should have an Amazon AWS account. Perform the following steps:

1. Create AWS user in AWS Console with following permissions:
   - AmazonEC2FullAccess
   - AmazonRoute53FullAccess
   - AmazonS3FullAccess
   - IAMFullAccess
   - AmazonVPCFullAccess
   Amazon documentation: Changing Permissions for an IAM User

2. Export AWS-specific environment variables:
   - AWS_ACCESS_KEY_ID
   - AWS_SECRET_ACCESS_KEY
   Amazon documentation: Environment Variables

Setting up Monarch Swarm Enterprise Server on Linux

A. Extract files from the Monarch Swarm Enterprise Server archive and configure server installation

1. Verify that the Ansible connection and the ./conf/inventory file are configured properly. Use the documentation for configuring AWS.
   Ansible Inventory

2. Review/edit server variables in ./conf/cluster.yaml file. You can also rename and edit the default ./conf/cluster.default.yaml file.
• Set the Kubernetes cluster name:

```
# AWS: Kubernetes cluster name
cluster_name: swarm.k8s.local
```

• Optional: Set vpc-id in vpc_id to create cluster in existing VPC:

```
# vpc_id to create cluster in existing VPC
#vpc_id: <some vpc id>
```

• Set cluster type, set aws in swarm_platform:

```
# aws, on-prem, standalone-linux
swarm_platform: aws
```

• Export AWS-specific environment variables. These properties should be the same as "Export AWS-specific environment variables" in the Requirements portion of this section:

```
$ export AWS_ACCESS_KEY_ID=Your Access Key ID
$ export AWS_SECRET_ACCESS_KEY=Your Secret Access Key
$ export AWS_DEFAULT_REGION=Your Default Region
```

3. To edit server configuration details, edit the file:

```
./playbooks/installers/aws/group_vars/all.yaml
```

B. Set up the server

**NOTE:** Make sure the all *.sh files are executable. To make files executable, run: $ chmod +x *.sh. All scripts should be run by the user ansible_user from ./conf/inventory. The default Ubuntu user is ubuntu.

To set up cluster run next script from the ./bin directory:

```
$ ./0-setup-server.sh
```

If ./0-setup-server.sh fails, fix the problem and re-run this command.

C. After installation

After installation, kubectl will be configured on current machine. You can verify the Kubernetes cluster connection by running kubectl cluster-info in the console.
STANDALONE INSTALLATION

This instruction will setup and configure Kubernetes cluster on single Kubernetes cluster. Only Linux installation is supported.

Requirements

- The server should be based on Ubuntu 16.04 distributive
- The server must meet the minimum requirements: 8 CPU, 16GB RAM

Setting up Monarch Swarm Enterprise Server on Linux

A. Extract files from the Monarch Swarm Enterprise Server archive and configure server installation:

1. Verify that the Ansible connection and the ./conf/inventory file are configured properly, Use the documentation for configuring Standalone Linux
   - Ansible Inventory
2. Review/edit server variables in the ./conf/cluster.yaml file. You can also rename and edit the default ./conf/cluster.default.yaml file.
   - Set cluster type, set standalone-linux in swarm_platform:
     ```bash
     # aws, on-prem, standalone-linux
     swarm_platform: standalone-linux
     ```
   - Specify additional hosts on which Kubernetes apiserver will be exposed for kubectl. If not present only from localhost, you can connect with kubectl to your cluster:
     ```bash
     apiserver_extra_sans: <external ip>
     ```

B. Setup the server

**NOTE:** Make sure the all *.sh files are executable. To make files executable, run: $ chmod +x *.sh.

To set up the cluster, run the next script from the ./bin directory:

```
$ ./0-setup-server.sh
```

If ./0-setup-server.sh fails, fix the problem and re-run this command.

C. After installation

After installation kubectl will be configured on the master node. You can use ssh to get access to the master node and execute kubectl cluster-info to verify the Kubernetes cluster installation.
ON-PREMISE INSTALLATION

These instructions will setup and configure Kubernetes cluster with an on-premise configuration. **Only Linux installation is supported.**

**Requirements**

- Master node and all workers nodes should be based on Ubuntu 16.04 distributive
- Master and worker nodes must meet the requirements presented in Requirements section

Setting Up Monarch Swarm Enterprise Server

**A. Extract files from the Monarch Swarm Enterprise Server archive and configure server installation:**

1. Verify that the Ansible connection and ./conf/inventory file are configured properly. Use the documentation for configuring On-Prem.

   Ansible inventory

2. Review/edit server variables in ./conf/cluster.yaml file. You can also rename and edit the default ./conf/cluster.default.yaml file.

   - Set cluster type by setting on-prem in swarm_platform:

     ```yaml
     # aws, on-prem, standalone-linux
     swarm_platform: on-prem
     ```

   - Specify additional hosts on which the Kubernetes apiserver will be exposed for kubectl. If not present only from localhost, you can connect with kubectl to your cluster:

     ```yaml
     apiserver_extra_sans: <external ip>
     ```

To change configuration details, use the documentation for On-Prem Configuration Details.

**B. Setup the Server**

**NOTE:** Make sure the all *.sh files are executable. To make files executable, run: 

```bash
chmod +x *.sh.
```

To set up the cluster, run the next script from the ./bin directory:

```bash
$ ./0-setup-server.sh
```

If ./0-setup-server.sh fails, fix the problem and re-run this command.
C. After Installation

After installation kubectl will be configured on the master node. You can use ssh to gain access to the master node and execute kubectl cluster-info to verify the Kubernetes cluster installation.

On-Prem Configuration Details

These commands should be executed from the project root directory.

These steps are automatically run via Ansible tasks to prepare master and worker nodes for Kubernetes cluster installation:

- Make sure the sudo has been configured. To check:
  ```
  $ sudo whoami
  ```
  and enter the sudo password.

- Python 2.7 must be installed:
  ```
  $ sudo apt install python
  ```

- The following ports should be opened for master and worker nodes:
  - 80/443 – for Monarch Swarm Enterprise Server application access
  - 22 – for Ansible access
  - 6443, 10250 – to gain access to Kubernetes API from kubectl and helm tools (optional)
  - 10.244.0.0/16 – flannel subnet
    ```
    $ firewall-cmd --zone=public --add-port=80/tcp --permanent
    $ firewall-cmd --zone=public --add-port=443/tcp --permanent
    $ firewall-cmd --zone=public --add-port=22/tcp --permanent
    $ firewall-cmd --zone=public --add-port=6443/tcp --permanent
    $ firewall-cmd --zone=public --add-port=10250/tcp --permanent
    $ firewall-cmd --zone=trusted --add-source=10.244.0.0/16 --permanent
    $ firewall-cmd --reload
    ```

  All ports should be opened for each master/worker in the cluster.

- Make sure docker is not installed. Uninstall older versions and Docker CE (if needed).

- Linux swap should be disabled on all machines. Remove Swap Space:
  - Disable the swap file
    ```
    $ sudo swapoff -a
    ```
  - Remove swap entry from the /etc/fstab file
  - Reboot machines
Pre-installation Configuration

For server configuration detail, edit file: `./playbooks/installers/on-prem/group_vars/all.yaml`

Swarm Pre-Installation Modules

KUBERNETES MODULES

Swarm modules are presented by additional preconfigured helm charts for Kubernetes cluster.

Modules:

<table>
<thead>
<tr>
<th>Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>helm</td>
<td>Kubernetes package manager for this interval.</td>
</tr>
<tr>
<td>dashboard</td>
<td>Kubernetes administration web interface</td>
</tr>
<tr>
<td>logging</td>
<td>ELK Stack log server with Kibana user interfaces</td>
</tr>
<tr>
<td>monitoring</td>
<td>Prometheus monitoring server with Grafana user interface</td>
</tr>
<tr>
<td>nfs</td>
<td>Internal NFS file storage. Configured as ReadWriteMany</td>
</tr>
<tr>
<td>hostpathprovisioner</td>
<td>Hostpath storage provisioner for Kubernetes</td>
</tr>
<tr>
<td>iscsi</td>
<td>iSCSI persistence values configuration</td>
</tr>
<tr>
<td>ceph-remote</td>
<td>Ceph-Remote storage provisioner for Kubernetes</td>
</tr>
<tr>
<td>fs-benchmark</td>
<td>Benchmarking file storage with FIO, Ioping</td>
</tr>
<tr>
<td>ingress</td>
<td>Nginx-ingress load balancer configuration</td>
</tr>
</tbody>
</table>

Configuration

**NOTE:** Do not use only one hostname for all *_ingress_host*.

To configure modules, edit the file `./conf/modules.yaml` (if the file does not exist, rename `modules.default.yaml`). To enable module installation, set `<module name>_enabled` to true.
1. Global module configuration

```yaml
---
# Modules
hostpath_enabled: false
ceph_remote_enabled: false
iscsi_enabled: false
nfs_enabled: false
fs_benchmark_enabled: false
ingress_enabled: false
dashboard_enabled: false
logging_enabled: false
monitoring_enabled: false
---
```

Possible configurations:

- **Local** - modules `hostpath`, `nfs`, `ingress`, `dashboard`
- **AWS** - modules `ingress`, `nfs`, `dashboard`, `logging`, `monitoring`
- **On-Prem (Ceph-remote)** - modules `ceph_remote`, `nfs`, `ingress`, `dashboard`, `logging`, `monitoring`
- **On-Prem (ISCSI)** - modules `iscsi`, `nfs`, `ingress`, `dashboard`, `logging`, `monitoring`
- **On-Prem (Hostpath)** - modules `hostpath`, `nfs`, `ingress`, `dashboard`, `logging`

2. Load balancer configuration

```yaml
---
# Load Balancer type
load_balancer_type: on-prem
---
```

Available types:

- **aws-single** - deploys a single instance of nginx-ingress to the cluster and uses ELB as entry point (the ELB host obtained should be passed to the `ingress_host` property for swarm-deploy)
- **on-prem** - deploys a single instance of nginx-ingress to the cluster in DaemonSet configuration and uses the host network.

3. Logging configuration

```yaml
# Host name of logging, e.g.: logs.swarm.local
logging_ingress_host: <logs host>
logging_recreate_certificate: true # update certificate
logging_namespace: logs
# certificate should be valid for <logs host>
logging_tls_key: "{{ lookup('env','PWD') }}/conf/tls.key"
logging_tls_crt: "{{ lookup('env','PWD') }}/conf/tls.crt"
```
To see more details, navigate to Logging configuration.

4. Monitoring configuration for Prometheus and Grafana

```yaml
# Host name of monitoring, e.g.: monitoring.swarm.local
monitoring_ingress_host: <monitoring host>
monitoring_recreate_certificate: true # update certificate
monitoring_namespace: monitoring
# certificate should be valid for <monitoring host>
monitoring_tls_key: "{{ lookup('env','PWD') }}/conf/tls.key"
monitoring_tls_crt: "{{ lookup('env','PWD') }}/conf/tls.crt"
```

To see more details, navigate to Monitoring configuration.

5. Kubernetes dashboard configuration

```yaml
# Host name of dashboard, e.g.: dashboard.swarm.local
dashboard_ingress_host: <dashboard host>
dashboard_recreate_certificate: true # update certificate
# certificate should be valid for <dashboard host>
dashboard_tls_key: "{{ lookup('env','PWD') }}/conf/tls.key"
dashboard_tls_crt: "{{ lookup('env','PWD') }}/conf/tls.crt"
```

To see more details, navigate to Kubernetes Dashboard configuration.

6. iSCSI persistence volume configuration

```yaml
# iSCSI target configuration
ISCSI_VOLUMES: []
- { name: volumeName, size: 10Gi, targetIp: iscsiIp, iqn: 'iqn',
  lun: lun }
- { name: volumeName, size: 10Gi, targetIp: iscsiIp, portals: [ip1, ip2], iqn: 'iqn',
  lun: lun, fsType: ext4, readonly: false,
  iscsiInterface: default }
```

To see more details, navigate to iSCSI configuration.

7. Ceph remote configuration

```yaml
# Ceph remote configuration
CEPH_ADMIN_SECRET_VALUE: ''
CEPH_USER_SECRET_VALUE: ''
CEPH_MONITORS: []
CEPH_ADMIN_ID: admin
CEPH_USER_ID: kube
CEPH_POOL_NAME: kube
```

To see more details, navigate to Ceph Remote configuration.
Setup Modules

**NOTE:** Make sure the all *.sh files are executable. To make files executable, run: `chmod +x *.sh`.

To install the modules, run the next script from the `./bin` directory:

```
$ ./l-setup-modules.sh
```

THE KUBERNETES DASHBOARD

**Component URL:** https://<dashboard_ingress_host>

Kubernetes Dashboard is a web-based Kubernetes user interface that is used to manage and deploy the Monarch Swarm Enterprise Server application and all of its components.

Opening the Kubernetes Dashboard:

- Open the URL of the Kubernetes Dashboard and set valid credentials.
- Provide a valid token and then sign in. There are two roles: admin, which has access to the whole cluster (all namespaces), and namespace user, which has access only to the selected namespace.

Obtaining Tokens

To get administrator and tenant tokens, execute `./utils.sh` from the `./bin` directory in the Monarch Swarm Enterprise Server installer and then select option 5. Alternatively, use kubectl on the remote server.

To get an admin token, run the following commands:

- For the on-premise configuration

  ```
  $ kubectl get secret -n kube-system $(kubectl get sa admin -n kube-system -o jsonpath='{.secrets[0].name}') -o jsonpath='{.data.token}' | base64 -d
  ```

- For the AWS configuration

  ```
  $ kubectl get secret -n kube-system $(kubectl get sa default -n kube-system -o jsonpath='{.secrets[0].name}') -o jsonpath='{.data.token}' | base64 -d
  ```
To get a `<namespace>` token, replace `<namespace>` with the tenant/namespace name and then run the following command:

```
$ kubectl get secret -n <namespace> $(kubectl get sa admin -n <namespace> -o jsonpath='{.secrets[0].name}') -o jsonpath='{.data.token}' | base64 -d
```

Your Kubernetes dashboard contains information about the whole cluster, including:

- Existing Namespaces (name, status, age)
- Instances or nodes (name, CPU, memory, age)
- Persistent volumes (name, capacity, access mode, status, age)
- Roles (name, type, age, namespace for this role)
- Storage classes (name, age, provisioner)

**LOGGING**

**Component URL:** https://<logs_ingress_host>

ELK Stack is used for aggregation and visualizing logs.

- **Kibana** is a tool to visualize log data
- **Elasticsearch** is a search and analytics data engine
- **Fluentd** is a data collector for unified logging layers

To use Kibana, you should define swarm index pattern `logstash-*`. This configuration is described in the documentation [Defining Your Index Patterns]. After configuring swarm pattern, you can work with Monarch Swarm Enterprise servers log on the Discover page.

By default, all logs (its time and source) from the whole cluster (from all namespaces) are displayed for the last 15 minutes. So, to see logs in move convenient view you can filter them by date or any available field and add any other log information. You can view detail information for all logs in Document data view.

**Web Application Logs**

To see logs from the web application in Kibana, you need:

- Set the following property in the env section to value "true" for test-swarm component: LOGGING_LOGSTASH_STDOUT. To see not only errors but also other logs (debug, info, warn), set the property LOGBACK_LOGLEVEL with value DEBUG.
Configure search by web application component, for example:

```json
{ "query": { "match": { "kubernetes.labels.app": { "query": "swarm","type": "phrase"}}}}
```

Add fields such as level, message, log to see logs in a more convenient view.

**NOTE:** You can view logs from any components of Monarch Swarm Enterprise Server. To do this you should add LOGGING_LOGSTASH_STDOUT property to this component and then in Kibana filter logs by the required component’s name.

Also, Kibana supports import of pre-defined objects, such as Dashboards, Searches, and Visualization. These objects can be imported as JSON files. To make them part of your Kibana, perform the following steps:

1. Open the Kibana URL.
2. Click the Management tab at the left-hand portion of the screen.
3. Click the **Saved Objects** link.
4. Click the **Import** button.
5. In the opened window, select the JSON file of your object and then click **Open**.
6. Click the **Yes, overwrite all objects** button in the popup that displays.

The following snippet is an example of a Search object that displays only logs from the 'dev' namespace saved in JSON format:

```json
[
    {
        "id": "9962f0b0-d91c-11e8-80c2-9d9ea8727842",
        "_type": "search",
        "_source": {
            "title": "Swarm Logs",
            "description": "",
            "hits": 0,
            "columns": [
                "kubernetes.container_name",
                "level",
                "message"
            ],
            "sort": [
                "@timestamp",
                "desc"
            ],
            "version": 1,
            "kibanaSavedObjectMeta": {
                "searchSourceJSON": "{"\n                    \"index\": \"91063290-d68f-11e8-80c2-9d9ea8727842\",\n                    \"highlightAll\": true,\n                    \"version\": true,\n                    \"query\": {\n                        \"query\": \"\",\n                        \"language\": \"lucene\"\n                    },\n                    \"filter\": {\n                        \"\": \n                        \"\n                    },\n                    \"meta\": {\n                        \"\": \n                    }
                }"
            }
        }
    }
]```
More information about importing Saved Objects is available in the [Kibana User Guide](#).

**MONITORING**

Prometheus and Grafana are used as the monitoring stack for Monarch Swarm Enterprise Server. Prometheus acts as the storage backend, and Grafana functions as the interface for analysis and visualization. Grafana supports a large number of plugins and features configured dashboards for Kubernetes servers. Prometheus and Grafana are installed with the Kubernetes modules.

**Access to the Grafana UI**

- **URL**: https://<monitoring_ingress_host>

Grafana credentials are stored in monitoring-grafana secret in the `<monitoring_namespace>` namespace. To get the username and password:

```bash
# password
$ kubectl get secret -n <monitoring_namespace> monitoring-grafana -o go-template='{{index .data "admin-password"}}' | base64 -d
# username
$ kubectl get secret -n <monitoring_namespace> monitoring-grafana -o go-template='{{index .data "admin-user"}}' | base64 -d
```

**Monitoring configuration**: ./conf/modules.yaml

- `<monitoring_namespace>` - monitoring namespace
- `<monitoring_path>` - monitoring path
Persistence Storage Configuration

CONFIGURING iSCSI

Configure iSCSI persistence volumes in ./conf/modules.yaml:

```
ISCSI_VOLUMES:
  - { # Required, name of the
    name: iscsi-1,
    volume, e.g.: iscsi-1
    size: 10Gi,
    # Required, capacity of the
    volume, e.g.: 172.31.xx.xx
    targetIp: 172.31.xx.xx,
    # Required, IP address of the
    iSCSI target, e.g.: 172.31.xx.xx
    portals: [172.31.xx.xx, 172.31.yy.yy], # Optional, array of
    additional IP addresses for iSCSI target, e.g.: [192.168.xx.xx, 192.168.yy.yy]
    iqn: 'iqn.2018-08.iscsi.xxxx:xxx.iscsi-1', # Required, IQN of the iSCSI
    target, should be in quotes e.g.: 'iqn.2005-01.iscsi.xxxx:xxx.iscsi-x'
    lun: 0, # Required, lun of the iSCSI
    target, e.g.: 5
    fsType: ext4, # Optional, volume file system
    type, _ext4_ is used by default, e.g.: xfs
    readonly: false, # Optional, Allows to mount
    volume as read only, _false_ by default, e.g.: true
    iscsiInterface: default # Optional, network interface
    of the iSCSI target, _default_ by default, e.g.: eth2
  }
  - { name: iscsi-2,
    size: 120Gi,
    targetIp: 172.31.xx.xx,
    iqn: 'iqn.2018-08.iscsi.xxxx:xxx.iscsi-2',
    lun: 2,
    fsType: xfs }
```

This module does not yet support authentication; i.m. volumes should be configured manually.

CONFIGURING CEPH REMOTE

Configure Ceph remote storage classes in ./conf/modules.yaml

```
CEPH_ADMIN_ID: admin # name of the admin ceph user, e.g.: admin.
CEPH_USER_ID: kube # name of the user who can use pool, e.g.: kubernetes.
CEPH_POOL_NAME: kube # name of the ceph pool, e.g.: kubernetes_storage.
CEPH_ADMIN_SECRET_VALUE: 'AQxxxxxxxlEmCxAAKS7tzZHSforkUE85xxxxxx==' # key for admin user, could be acquired by the following command: 'sudo ceph --cluster ceph auth get-key client.<admin>'
CEPH_USER_SECRET_VALUE: 'AQxxxxxYsXNUNBAAMTEW1/WnzXdmDZIBxxxxxx==' # key for user, could be acquired by the following command: 'sudo ceph --cluster ceph auth get-key client.<user>'
CEPH_MONITORS: # list of the ceph monitors to connect to, domain name or IP with port number, e.g.: ['ceph.xxx.xxx:6789', '10.10.xx.xx:6789']
  - monitor-address=x:6789
```
Monarch Swarm Enterprise Server Installation

See full requirements description for setting up Monarch Swarm Enterprise Server Swarm in Monarch Swarm Enterprise Server Requirements.

MONARCH SWARM INSTALLATION STEPS

1. Put license.lic file into the .conf folder of the extracted directory.

2. Generate and copy to .conf the certificate tls.crt and private key tls.key for the domain from ingress_host. The Swarm Enterprise supports wildcard certificates.

3. Review/edit server variables in this file .conf/swarm.yaml, or rename and edit .conf/swarm.default.yaml file.
   - Configure helm presets, e.g.:
     ```yaml
     helm_config_presets: [common, aws, filesystem-nfs]
     ```

     Possible values:
     - common - basic helm configuration
     - common-on-prem - on-prem helm configuration
     - common-single-server - linux-standalone helm configuration
     - filesystem-static - configuration for pre-configured static volumes
     - filesystem-nfs - nfs helm configuration - provide ReadWriteMany accessMode (required for all installation types)
     - secrets - docker secret
     - aws - AWS helm configuration
     - s3 - AWS S3 helm configuration for ReadWriteMany
     - sso - SSO configuration
     - repository - private docker registry configuration
- Specify the public domain name on the Kubernetes master node. **This property is not used and should not be set for aws platform.** The domain name should be resolved by DNS.

  ```yaml
  # Public host, only for `on-prem` and `standalone-linux`, e.g.: example.com:
  ingress_host: "<server host>"
  ```

- Set username from the remote server. This user should be valid in the machine on which the application will be installed. The default Ubuntu user is **ubuntu**.

  ```yaml
  ansible_remote_user: ubuntu
  ```

- Pass valid docker credentials (provided by Datawatch):

  ```yaml
  docker_username: <docker user>
  docker_password: <docker password>
  ```

- The version of the Monarch Swarm Enterprise Server Docker images is set automatically, version can be changed to update the Monarch Swarm Enterprise Server application.

  ```yaml
  swarm_image_version: 2.2-491-2f8f859f15aa
  ```

- Set Swarm tenant name.

  ```yaml
  namespace: swarm
  ```

- Configure the certificate for the domain **ingress_host**, by default.

  ```yaml
  tls_key: "{{ lookup('env','PWD') }}/conf/tls.key"
  tls_crt: "{{ lookup('env','PWD') }}/conf/tls.crt"
  ```

- Optional, set path to the Monarch Swarm Enterprise Server license, by default:

  ```yaml
  license_file_path: "{{ lookup('env','PWD') }}/conf/license.lic"
  ```

To specify Monarch Swarm Enterprise Server configurations, navigate to

- Application Properties
- S3 File Library Configuration
- SSO Configuration
Deploying the Application

**NOTE:** Make sure the all *.sh files are executable. To make files executable, run: $ chmod +x *.sh.

To set up the cluster, run the next script from the ./bin directory:

$ ./2-deploy-swarm.sh

If $ ./2-deploy-swarm.sh fails, fix the problem and re-run this command.

After Installation

The installer sets up the required services.

Access to the Monarch Swarm application is done via the URL: https://<master host>/.

**SWARM CLEANUP IN CASE OF INSTALLATION ERRORS**

In case of issues during Monarch Swarm installation, the following actions should be taken to delete it and make a new installation attempt.

Find out the name of the Swarm helm package. This could be done via command line or by checking the ./conf/swarm.yaml file.

Check the value of the namespace variable in ./conf/swarm.yaml. Alternatively, execute the following commands in Shell and find the name of the deployment where the chart is swarm-0.1.0:

```bash
$ helm ls
NAME            REVISION        UPDATED                         STATUS
CHART                           NAMESPACE
--------------------------------------
dashboard       1               Mon Sep 17 09:16:48 2018  DEPLOYED
kubernetes-dashboard-0.7.3 kube-system
DEPLOYED dev 1               Mon Sep 17 09:53:50 2018  DEPLOYED
swarm-0.1.0                      dev
DEPLOYED elk 2               Mon Sep 17 09:43:53 2018  DEPLOYED
elk-0.1.0 logs
DEPLOYED heapster 1 Mon Sep 17 09:16:50 2018  DEPLOYED
heapster-0.3.1 kube-system
DEPLOYED lb 1               Mon Sep 17 09:16:09 2018  DEPLOYED
nginx-ingress-0.20.3 kube-system
DEPLOYED monitoring 2 Mon Sep 17 09:45:30 2018  DEPLOYED
monitoring-0.1.0 monitoring
nfs 3               Tue Sep 18 12:33:58 2018  DEPLOYED
nfs-server-provisioner-0.2.1 kube-system
```

In the example above, the name is dev.
Delete the deployments and namespace (the same as swarm deployment name) by using the commands below:

```
$ helm delete --purge <swarm deployment name>
```

Delete the namespace and all volumes:

```
$ kubectl delete namespace <swarm namespace>
```

**NOTE:** Your list of helm deployments may not contain dev-lb as in the example above. In this case, you may safely ignore helm error `Error: release: "dev-lb" not found.`

## CONFIGURING SINGLE SIGN-ON (SSO)

To enable SSO in Monarch Swarm Enterprise Server:

1. Configure `./conf/swarm.yaml`
   - Add sso to `helm_config_presets`:
     - for AWS configuration
       ```yaml
       helm_config_presets: [common, aws, filesystem-nfs, sso]
       ```
     - for On-Prem configuration:
       ```yaml
       helm_config_presets: [common, common-on-prem, sso]
       ```
   - Enable SSO and set path to configuration files:
     ```yaml
     # SSO & Kerberos configuration, disabled by default
     sso_enable: true
     sso_krb5_conf: "{{ lookup('env','PWD') }}/conf/krb5.conf"
     sso_keytab: "{{ lookup('env','PWD') }}/conf/linuxsso.keytab"
     ```
   - Set DNS server for Kubernetes, e.g.:
     ```yaml
     sso_upstream_nameservers: "["172.31.65.246"]"
     ```

2. Create Kerberos configuration file `./conf/krb5.conf` or rename and edit `./conf/krb5.sample.conf`, e.g.:

   ```yaml
   [libdefaults]
   default_realm = DWTEST.TEST
   default_keytab_name = /keytab/linuxsso.keytab
   forwardable=true
   dns_lookup_realm = true
   rdns = false
   dns_lookup_kdc = true
   ```
[realms]
DWTEST.TEST = {
    kdc = WIN-SERVER.dwtest.test:88
    admin_server = WIN-SERVER.dwtest.test:88
}

[domain_realm]
dwtest.test=DWTEST.TEST
.dwtest.test=DWTEST.TEST

[appdefaults]
kinit = {
    renewable = true
    forwardable= true
}

NOTE: Do not modify keytab path - default_keytab_name.

3. Generate ./conf/linuxsso.keytab or rename and edit ./conf/linuxsso.default.keytab. Use the documentation in ktpass | Microsoft Docs All You Need to Know About Keytab Files as a guide.

4. Configure ./helm/values/swarm/sso.yaml

```
### This is the example of the SSO configuration.
## - For use external DNS in the Kubernetes please set 'sso_upstream_nameservers' in the './conf/swarm.yaml'.

core-api:
    config:
        java:
            ## Include Kerberos configuration in JAVA_OPTS for Swarm application
            otherOpts: -Djava.security.krb5.conf=./sso/krb5.conf
            authentication:
                provider: ldap
                ldap:
                    ## LDAP domain name
                    domain: DWTEST.TEST
                    ## LDAP server url
                    server: ldap://WIN-SERVER.dwtest.test/
                    ## LDAP search base for users
                    searchBase: dc=dwtest,dc=test
                    ## LDAP filter
                    searchFilter: (|(userPrincipalName={0})
                    (sAMAccountName={0}))
                userRoles:
                    - 2
```
sso:
  enabled: true
  servicePrincipal: HTTP/xxxxxx.compute-1.amazonaws.com@DWTEST.TEST

## Keytab path with user principals, this property is also configured in krb5.conf (`default_keytab_name`)
  keyTabLocation: /keytab/linuxsso.keytab
  requestRegex: ^/api/.*/ldap_sso

krb5:
  ## Name of the Kubernetes configmap with krb5.conf file, should be the same as the value in playbooks/swarm-deploy/roles/sso/vars/main.yaml:3 (`sso_krb5_name`)
  name: krb5-conf
  mountPath: /sso

keytab:
  ## Name of the Kubernetes security with linuxsso.keytab file, should be the same as the value in playbooks/swarm-deploy/roles/sso/vars/main.yaml:3 (`sso_keytab_name`)
  name: krb5-keytab
  mountPath: /keytab

5. After `./helm/values/swarm/sso.yaml` configuration, execute `./2-deploy-swarm.sh` from the `.bin` directory.

## S3 FILE LIBRARY CONFIGURATION

To enable S3 file libraries in Monarch Swarm Enterprise Server:

1. Configure `swarm.yaml`
   - Add `s3` to `helm_config_presets`:
     - for On-Prem configuration
       ```yaml
       helm_config_presets: [common, common-on-prem, s3]
       ```
     - for AWS configuration:
       ```yaml
       helm_config_presets: [common, aws, filesystem-nfs, s3]
       ```

2. Configure `./helm/values/swarm/s3.yaml`:

```
#####################################################
## S3 File-library configuration
## This is the example of the S3 configuration.
#####################################################

## S3 configuration
config:
  optionalEnv:
```
## Required:

- **name**: `SPRING_LIQUIBASE_PARAMETERS_S3ACCESSKEY`  
  value: "<aws_access_key_id>"

## Required:

- **name**: `SPRING_LIQUIBASE_PARAMETERS_S3PRIVATEKEY`  
  value: "<aws_secret_access_key>"

## Required:

- **name**: `SPRING_LIQUIBASE_PARAMETERS_S3BUCKET`  
  value: "<s3_bucket_name>"

- **name**: `SPRING_LIQUIBASE_PARAMETERS_S3PATH`  
  value: "<s3_bucket_path>"

3. After `./helm/values/swarm/s3.yaml` configuration, execute `./2-deploy-swarm.sh` from the `./bin` directory.

## APPLICATION PROPERTIES

### Common Server Configuration

Configure CPU and memory resources in the file `./helm/values/swarm/common.yaml`.

```yaml
# The default configuration for a cluster with 3 working nodes. Each working node has 8 CPU, 32Gb RAM.
# default:
- &storageClass "" # default persistent storage class

core-api:
  replicaCount: 2
  image:
    pullSecrets: &imagePullSecrets regsecret
  config:
    java:
      minHeapSize: 512M
      maxHeapSize: 5G
    logging:
      logstash:
        enabled: &logstashEnabled true # enable logging in json
    resources:
      requests: # requested cpu and memory resources for the Monarch Swarm Enterprise Server application
        cpu: 1
        memory: 2.6Gi
      limits: # limit cpu and memory resources for the Monarch Swarm Enterprise Server application
        cpu: 4
        memory: 5.6Gi
```
data-engine-api:
  replicaCount: 2
  image:
    pullSecrets: *imagePullSecrets
  config:
    java:
      minHeapSize: 512M
      maxHeapSize: 4G
  logging:
    logstash:
      enabled: *logstashEnabled # enable logging in json
  resources:
    requests: # requested cpu and memory resources for the Monarch Swarm Enterprise Server application
      cpu: 1
      memory: 2.6Gi
    limits: # limit cpu and memory resources for the Monarch Swarm Enterprise Server application
      cpu: 3
      memory: 4.3Gi

data-engine-api-postgres:
  imagePullSecrets: *imagePullSecrets
  persistence:
    storageClass: *storageClass
    size: 32Gi # default size of PostgreSQL database persistent storage

data-engine-worker:
  replicaCount: 2
  image:
    pullSecrets: *imagePullSecrets
  config:
    java:
      minHeapSize: 512M
      maxHeapSize: 5G
  logging:
    logstash:
      enabled: *logstashEnabled # enable logging in json
  resources:
    requests: # requested cpu and memory resources for the Monarch Swarm Enterprise Server application
      cpu: 1.5
      memory: 2.6Gi
    limits: # limit cpu and memory resources for the Monarch Swarm Enterprise Server application
      cpu: 3
      memory: 5.3Gi

data-engine-batch:
  replicaCount: 1
  image:
    pullSecrets: *imagePullSecrets
  config:
    java:
      minHeapSize: 512M
      maxHeapSize: 4G
logging:
  logstash:
    enabled: *logstashEnabled # enable logging in json
resources:
  requests: # requested cpu and memory resources for the Monarch Swarm Enterprise Server application
    cpu: 1
    memory: 1.6Gi
  limits: # limit cpu and memory resources for the Monarch Swarm Enterprise Server application
    cpu: 3
    memory: 4.3Gi
depostgres:
  imagePullSecrets: *imagePullSecrets
  persistence:
    storageClass: *storageClass
    size: 320Gi # allocated size for Monarch Swarm Enterprise Server batch data engine storage
resources:
  requests: # requested cpu and memory resources for the Monarch Swarm Enterprise Server batch data engine
    cpu: 1
    memory: 4Gi
  limits: # limit cpu and memory resources for the Monarch Swarm Enterprise Server batch data engine
    cpu: 5
    memory: 10Gi
core-api-postgres:
  imagePullSecrets: *imagePullSecrets
  persistence:
    storageClass: *storageClass
    size: 32Gi # default size of PostgreSQL database persistent storage
depostgres:
  imagePullSecrets: *imagePullSecrets
  persistence:
    storageClass: *storageClass
    size: 320Gi # allocated size for Monarch Swarm Enterprise Server data engine storage
resources:
  requests: # requested cpu and memory resources for the Monarch Swarm Enterprise Server data engine
    cpu: 2
    memory: 4Gi
  limits: # limit cpu and memory resources for the Monarch Swarm Enterprise Server data engine
    cpu: 6
    memory: 12Gi
redis:
  image:
    pullSecrets:
      - *imagePullSecrets
  master:
    persistence:
storageClass: *storageClass  # set storage for Redis server

cassandra:
  image:
    pullSecrets: *imagePullSecrets
  config:  # Cassandra memory configuration
    max_heap_size: 2253M
    heap_new_size: 512M
  resources:  # requested cpu and memory resources for Cassandra
    requests:
      memory: 4Gi
      cpu: 2
    limits:  # limit cpu and memory resources for the Cassandra database
      memory: 4Gi
      cpu: 2
  persistence:
    size: 32Gi  # allocated size for Cassandra storage
  storageClass: *storageClass

machine-learning:
  image:
    pullSecrets: *imagePullSecrets
  persistence:
    storageClass: *storageClass
  resources:
    requests:  # requested cpu and memory resources for the Machine learning service
      cpu: 1
      memory: 1Gi
    limits:  # limit cpu and memory resources for the Machine learning service
      cpu: 2
      memory: 3.6Gi
  config:
    logging:
      logstash:
        enabled: *logstashEnabled
  spark:
    resources:
      requests:  # requested cpu and memory resources for the Spark application
        cpu: 0.5
        memory: 4Gi
      limits:  # limit cpu and memory resources for the Spark application
        cpu: 1
        memory: 4Gi
  config:
    logging:
      logstash:
        enabled: *logstashEnabled

**NOTE:** Properties for setup persistence size should be defined before installation and cannot be changed later.
Application Properties

Monarch Swarm Enterprise Server application properties can be overridden using Kubernetes Dashboard. You can override all properties that are set in application-prod.yml.

A. Properties for core-api Deployment

```
"env": [
    {
      "name": "JAVA_OPTS",
      "value": "-Xms5G -Xmx5G"
    },
    {
      "name": "APPLICATION_SECURITY_AUTHENTICATION_PROVIDER",
      "value": "basic"
    },
    {
      "name": "APPLICATION_LICENSE_FILEPATH",
      "value": "/license/license.lic"
    },
    {
      "name": "ProgramData",
      "value": "/var/swarm/file-library"
    },
    {
      "name": "SPRING_DATASOURCE_URL",
      "value": "jdbc:postgresql://dev-core-api-postgres:5432/newserver?user=newserver&password=newserver&loginTimeout=30"
    },
    {
      "name": "APPLICATION_DATA_ENGINE_API_URL",
      "value": "http://dev-data-engine-api:80"
    },
    {
      "name": "SPRING_DATASOURCE_HIKARI_MAXIMUM_POOL_SIZE",
      "value": "15"
    },
    {
      "name": "APPLICATION_METRICS_PROMETHEUS_ENABLED",
      "value": "true"
    },
    {
      "name": "SWARM_SLEEP",
      "value": "0"
    },
    {
      "name": "LOGGING_LOGSTASH_STDOUT",
      "value": "true"
    },
    {
      "name": "SPRING_DATA_CASSANDRA_ENABLED",
      "value": "true"
    }
]
```
B. Properties for data-engine-api Deployment

"env": [
  {
    "name": "JAVA_OPTS",
    "value": "-Xms4G -Xmx4G"
  },
  {
    "name": "SPRING_PROFILES_ACTIVE"
  },
  {
    "name": "SPRING_RABBITMQ_DYNAMIC",
    "value": "true"
  },
  {
    "name": "SPRING_RABBITMQ_HOST",
    "value": "dev-rabbitmq-ha"
  },
  {
    "name": "SPRING_RABBITMQ_PORT",
    "value": "5672"
  },
  {
    "name": "SPRING_RABBITMQ_USERNAME",
    "value": "true"
  },
  {
    "name": "SPRING_RABBITMQ_PASSWORD",
    "value": "dev-rabbitmq-ha"
  }
]
"value": "rabbitmq"
},
{
 "name": "SPRING_RABBITMQ_PASSWORD",
 "value": "rabbitmq"
},
{
 "name": "ProgramData",
 "value": "/var/swarm/file-library"
},
{
 "name": "SPRING_DATASOURCE_URL",
},
{
 "name": "APPLICATION_DATA_ENGINE_LOCAL",
 "value": "false"
},
{
 "name": "APPLICATION_DATA_ENGINE_POSTGRES_ENABLED",
 "value": "false"
},
{
 "name": "APPLICATION_DATA_ENGINE_POSTGRES_BY_DEFAULT",
 "value": "false"
},
{
 "name": "APPLICATION_METRICS_PROMETHEUS_ENABLED",
 "value": "true"
},
{
 "name": "APPLICATION_IOPLACE_NATIVE_LIBS_TO_TEMPORARY_FOLDER",
 "value": "true"
},
{
 "name": "APPLICATION_CORE_API_URL",
 "value": "http://dev-core-api:80"
},
{
 "name": "SPRING_DATA_CASSANDRA_ENABLED",
 "value": "true"
},
{
 "name": "SPRING_DATA_CASSANDRA_CONTACT_POINTS",
 "value": "dev-cassandra"
},
{
 "name": "SWARM_SLEEP",
 "value": "0"
},
{
 "name": "LOGGING_LOGSTASH_STDOUT",
 "value": "true"
C. Properties for data-engine-worker Statefulset

"env": [
  {
    "name": "JAVA_OPTS",
    "value": "-Xms5G -Xmx5G"
  },
  {
    "name": "SERVER_PORT",
    "value": "8082"
  },
  {
    "name": "SPRING_PROFILES_ACTIVE",
    "value": "live"
  },
  {
    "name": "SPRING_RABBITMQ_DYNAMIC",
    "value": "true"
  },
  {
    "name": "SPRING_RABBITMQ_HOST",
    "value": "dev-rabbitmq-ha"
  },
  {
    "name": "SPRING_RABBITMQ_PORT",
    "value": "5672"
  }
]
```json
{
  "name": "SPRING_RABBITMQ_USERNAME",
  "value": "rabbitmq"
},
{
  "name": "SPRING_RABBITMQ_PASSWORD",
  "value": "rabbitmq"
},
{
  "name": "ProgramData",
  "value": "/var/swarm/file-library"
},
{
  "name": "APPLICATION_DATA_ENGINE_POSTGRES_URL",
  "value": "jdbc:postgresql://dev-postgres:5432/postgres?user=postgres&password=postgres&loginTimeout=30"
},
{
  "name": "APPLICATION_DATA_ENGINE_POSTGRES_HIKARI_MAXIMUM_POOL_SIZE",
  "value": "15"
},
{
  "name": "APPLICATION_DATA_ENGINE_STORE_PARALLELISM",
  "value": "10"
},
{
  "name": "APPLICATION_DATA_ENGINE_STORE_CACHE_DATA_SOURCE_ON_UPDATE",
  "value": "true"
},
{
  "name": "APPLICATION_METRICS_PROMETHEUS_ENABLED",
  "value": "true"
},
{
  "name": "APPLICATION_IO_PLACE_NATIVE_LIBS_TO_TEMPORARY_FOLDER",
  "value": "true"
},
{
  "name": "APPLICATION_CORE_API_URL",
  "value": "http://dev-core-api:80"
},
{
  "name": "SPRING_DATA_CASSANDRA_ENABLED",
  "value": "true"
},
{
  "name": "SPRING_DATA_CASSANDRA_CONTACT_POINTS",
  "value": "dev-cassandra"
},
{
  "name": "SWARM_SLEEP",
  "value": "0"
}
}```
D. Properties for data-engine-batch Deployment

"env": [
  {
    "name": "JAVA_OPTS",
    "value": "-Xms512M -Xmx4G"
  },
  {
    "name": "SERVER_PORT",
    "value": "8082"
  },
  {
    "name": "SPRING_PROFILES_ACTIVE",
    "value": "batch"
  },
  {
    "name": "SPRING_RABBITMQ_DYNAMIC",
    "value": "true"
  },
  {
    "name": "SPRING_RABBITMQ_HOST",
    "value": "dev-rabbitmq-ha"
  },
  {
    "name": "SPRING_RABBITMQ_PORT",
    "value": "5672"
  }
]
{,
  "name": "SPRING_RABBITMQ_USERNAME",
  "value": "rabbitmq"
},
{ "name": "SPRING_RABBITMQ_PASSWORD",
  "value": "rabbitmq"
},
{ "name": "ProgramData",
  "value": "/var/swarm/file-library"
},
{ "name": "APPLICATION_DATA_ENGINE_POSTGRES_URL",
  "value": "jdbc:postgresql://localhost:5432/postgres?user=postgres&password=postgres&loginTimeout=30"
},
{ "name": "APPLICATION_DATA_ENGINE_POSTGRES_HIKARI_MAXIMUM_POOL_SIZE",
  "value": "15"
},
{ "name": "APPLICATION_DATA_ENGINE_STORE_PARALLELISM",
  "value": "10"
},
{ "name": "APPLICATION_DATA_ENGINE_STORE_CACHE_DATA_SOURCE_ON_UPDATE",
  "value": "true"
},
{ "name": "APPLICATION_METRICS_PROMETHEUS_ENABLED",
  "value": "true"
},
{ "name": "APPLICATION_IO_PLACE_NATIVE_LIBS_TO_TEMPORARY_FOLDER",
  "value": "true"
},
{ "name": "APPLICATION_CORE_API_URL",
  "value": "http://dev-core-api:80"
},
{ "name": "SPRING_DATA_CASSANDRA_ENABLED",
  "value": "true"
},
{ "name": "SPRING_DATA_CASSANDRA_CONTACT_POINTS",
  "value": "dev-cassandra"
},
{ "name": "SWARM_SLEEP",
  "value": "0"
Setting Descriptions

- JAVA_OPTS - Java opts for Monarch Swarm Enterprise Server application
- APPLICATION_LICENSE_FILEPATH - path for the license file
- ProgramData - path for file library
- SPRING_DATASOURCE_URL - connection to the Monarch Swarm Enterprise Server database
- SPRING_RABBITMQ_LISTENER_DIRECT_CONSUMERS_PER_QUEUE - Number of consumers per queue.
- APPLICATION_DATA_ENGINE_POSTGRES_URL - connection to the Monarch Swarm Enterprise Server data engine
- APPLICATION_DATA_ENGINE_POSTGRES_HIKARI_MAXIMUM_POOL_SIZE - database pool size for the Monarch Swarm Enterprise Server data engine
- APPLICATION_DATA_ENGINE_STORE_PARALLELISM - parallel store configuration for the Monarch Swarm Enterprise Server data engine
- APPLICATION_DATA_ENGINE_STORE_CACHE_DATA_SOURCE_ON_UPDATE - enable cache on update functionality
- SPRING_DATASOURCE_HIKARI_MAXIMUM_POOL_SIZE - maximum pool size for data engine
- LOGGING_LOGSTASH_STDOUT - enable logstash integration
- SPRING_DATA_CASSANDRA_ENABLED - enable Cassandra
- SPRING_DATA_CASSANDRA_CONTACT_POINTS - Cassandra configuration
- SPRING_REDIS_ENABLED - enable Redis cache
- SPRING_REDIS_URL - connection to the Redis server
- APPLICATION_DATA_ENGINE_STORE_OBJECTS_FACTORY - object factory configuration for the Monarch Swarm Enterprise Server data engine
- APPLICATION_DATA_ENGINE_STORE_LOCKS - store locks configuration for the Monarch Swarm Enterprise Server data engine
- SPRING_CACHE_JCACHE_PROVIDER - cache provider for the Monarch Swarm Enterprise Server application
- APPLICATION_SERVER_INTERNET_ADDRESS - external URL for the Monarch Swarm Enterprise Server application
- SPRING_HTTP_MULTIPART_MAXFILESIZE - the maximum size of files that may be uploaded to the application, the default value is 2000MB
- SPRING_HTTP_MULTIPART_MAXREQUESTSIZE - meaning total request size for a multipart/form-data
- APPLICATION_HTTP_CACHE_TIMETOLIVEINDAYS - the amount of time in days that may lapse before a data source’s cache times out, the default value is 31
- APPLICATION_DATA_ENGINE_STORE_EXPORT_DATA_AWAIT_TIMEOUT_IN_SEC - the number of seconds that should pass before an export times out, the default value is 3600
- APPLICATION_DATA_ENGINE_STORE_DESIgn_MODE_LIMIT - Describes the row limit to be used for data sources in Design Mode, the default value is 100000
- APPLICATION_DATA_ENGINE_SUGGESTION_PREPARE_CRON - settings for jobs that calculate suggestions based on data type and content, the default value is 0 0 1 * * *
- APPLICATION_DATA_ENGINE_SUGGESTION_RANK_THRESHOLD - settings for suggestions based on data type and content; shows minimum rank for retrieving and sorting suggestions, the default value is 0.2
- APPLICATION_DSL_SOURCE_CLEANER_CRON - the schedule for the job which deletes temporary sources, the default value is 0 0 * * * MON-FRI
- APPLICATION_DSL_SOURCE_EXPIRATION_IN_HOURS - the number of hours after which the temporary source can be deleted by the job, the default value is 24
- APPLICATION_DSL_TEMPORARY_ITEM_CLEANER_CRON - the schedule for the job which deletes temporary objects, the default value is 0 0 * * * MON-FRI
- APPLICATION_DSL_TEMPORARY_ITEM_EXPIRATION_IN_HOURS - the number of hours after which the temporary object can be deleted by the job, the default value is 24
- APPLICATION_IO_WRITER_INTERNAL_CASSANDRA_BATCH_SIZE - the batch size for exports to Cassandra, the default value is 10
- APPLICATION_IO_WRITER_INTERNAL_CASSANDRA_NUM_FUTURES - the number of futures used for export, the default value is 1000
- APPLICATION_IO_WRITER_DATABASE_BATCH_SIZE - the batch size for exports to SQL, Oracle, and PostgreSQL, the default value is 24
- APPLICATION_IO_WRITER_WATSON_HTTP_CLIENT_TIMEOUT - the time in seconds that may elapse before connections to IBM Watson Analytics time out, the default value is 300
- APPLICATION_IO_WRITER_COGNOS_HTTP_CLIENT_TIMEOUT - the time in seconds that may elapse before connections to IBM Cognos Analytics time out, the default value is 600
- APPLICATION_IO_READER_PREVIEW_LIMIT - the row limit for previewing data sources, the default value is 1000
APPLICATION_SECURITY_AUTHENTICATION_XAUTH_TOKENVALIDITYINSECONDS - the number of seconds should elapse before a user times out, the default value is 1800

NOTE: The name of all properties must be written in Upper Case.

After adding/editing properties, the corresponding deployment will be restarted automatically and edited properties will be applied.

CREATING NEW TENANTS

When you create new tenant in Kubernetes, a new namespace will be created and a new Monarch Swarm Enterprise Server application will be deployed to the new tenant of the cluster.

• Create a new Swarm configuration (e.g: new-tenant) ./conf/new-tenant.yaml file from swarm.default.yaml. The file ./conf/new-tenant.yaml should be configured as the default ./conf/swarm.yaml. Set in namespace name of the new tenant:

    namespace: new-tenant

• Set the host url for the new tenant:

    ingress_host: <new tenant server host>

    If you want the same configuration as default tenant, clone swarm.yaml with the new-tenant.yaml name and set the new namespace name and ingress host.

• Run script for Monarch Swarm Enterprise Server deployment from the ./bin directory:

    $ ./2-deploy-swarm.sh new-tenant

You can create several tenant configuration files. If you run ./2-deploy-swarm.sh without a parameter, ./conf/swarm.yaml is used.

NOTE: For AWS configurations, a new load balancer (AWS ELB) will be used for each tenant. For On-Prem configurations, a new load balancer should be defined manually in ingress.
Server Maintenance

This section describes how to maintain various components of Monarch Swarm Enterprise Server.

Utilities

Utilities provide help commands for server maintenance. All commands should be run from the ./bin directory.

It is possible to use a different cluster configuration file instead of swarm.yaml. To do so, add the configuration file name without the extension in the ./conf directory. For example:

```
$ ./utils.sh swarm_uat
```

- List all libraries from the remote server. These libraries are included in the classpath of the Monarch Swarm Enterprise Server application:
  
  `$ ./utils.sh # option 1`

- Download libraries from the remote server. All libraries will be copied to the ./libs directory:
  
  `$ ./utils.sh # option 2`

- Upload libraries from the ./libs directory. If the ./libs directory doesn't exist in artifact root folder you should create one and place all needed libraries into this folder. To simplify the process, run libs-download utility first, change libraries in the ./libs folder (add, delete) and then perform the following:
  
  `$ ./utils.sh # option 3`

- Get secret tokens for the cluster and for tenant administrators:
  
  `$ ./utils.sh # option 5`

- License update:
  
  `$ ./utils.sh # option 6`

- Stop the Monarch Swarm Enterprise Server application:
  
  `$ ./utils.sh # option 7`

- Start the Monarch Swarm Enterprise Server application:
  
  `$ ./utils.sh # option 8`

- Restart the Monarch Swarm Enterprise Server application:
  
  `$ ./utils.sh # option 9`
Get a post-installation report:

```
$ ./utils.sh # option 10
```

Get a Monarch Swarm post-installation validation check:

```
$ ./utils.sh # option 11
```

## Maintaining Monarch Swarm Enterprise Server

Monarch Swarm Enterprise Server is built on the Kubernetes cluster. For maintaining this server, the following tools can be used:

- kubectl
- helm
- ark

To configure these tools, download the Kubernetes config file from Monarch Swarm Enterprise Server `$HOME/.kube/config` and register the environment variable `KUBECONFIG`, e.g.:

```
export KUBECONFIG=$HOME/config
```

After setting up `KUBECONFIG`, all of the related tools (i.e., kubectl, helm, ark) will work with the Monarch Swarm Enterprise server correctly.

More details about these tools and configuring access for your cluster can be found in the official guides.

**Additional Information**

- [Organizing Cluster Access and kubeconfig](#)
Upgrading Monarch Swarm Enterprise Server

These instructions do not support Kubernetes cluster upgrades.

The Monarch Swarm Enterprise Server application uses helm upgrade, which is configured as an Ansible task. The Monarch Swarm Enterprise Server handles the changing Docker images version, downloads new Docker images from the private registry, and runs them.

To update cluster configuration and the Monarch Swarm Enterprise Server version, you need to download the new version of Monarch Swarm Enterprise Server. Update swarm.yaml with your settings in the ./conf directory. If you modified the helm configuration in ./helm/values/swarm/, you need to copy/merge this configuration in the new installer. After merge configuration, run the ./2-deploy-swarm.sh script from ./bin to update Monarch Swarm Enterprise Server with the new version and your configurations.

To update certificates and license use the same commands as for the Monarch Swarm Enterprise Server Installation. Just update license and certificate files in the ./conf folder and run ./2-deploy-swarm.sh from ./bin.

Additional Information

- Helm upgrade
- Upgrades - Kubernetes
- kubeadm Upgrade - Kubernetes

Backing Up and Restoring the Server

For server backup, use Heptio Ark, a utility for managing disaster recovery, for your Kubernetes cluster resources and persistent volumes.

Ark is capable of backing up only cluster resources for Monarch Swarm Enterprise installation. Backup of persistent data should be manually performed by the cluster administrator.

PREREQUISITES

The following steps must be performed prior to backing up your server:

1. Configure backups-server variables in ./conf/swarm.yaml.
• Set a valid S3 bucket name `aws_s3_bucket` that will be used for backup storage

• Set valid AWS credentials `aws_access_key_id` and `aws_secret_access_key`

```bash
# Backup configuration
aws_s3_bucket:
aws_access_key_id:
aws_secret_access_key:
```

2. Configure the server connection `ansible_host` in `./conf/inventory`.

**SETTING UP THE ARK SERVER**

Run playbook with backup-server installation for Monarch Swarm Enterprise Server from the `./bin` directory

```
$ ./backup.sh
```

**INSTALLING THE CLIENT**

For this example, we recommend downloading a pre-built release. Make sure that you install the client somewhere in your `$PATH`. The Ark command line tool and `kubeconfig` should be installed. All commands for backup and restore are described in the Ark User Guide.

**BACK UP**

Create a backup for any object that matches the app=swarm label selector:

```
$ ark backup create <backup-name> -l release=<release-name>
```

**RESTORATION**

Run:

```
$ ark restore create --from-backup <backup-name>
```

Run:

```
$ ark restore get
```
After restoration is completed, the following output is obtained:

<table>
<thead>
<tr>
<th>NAME</th>
<th>BACKUP</th>
<th>STATUS</th>
<th>WARNINGS</th>
<th>ERRORS</th>
<th>CREATED</th>
<th>SELECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-backup-20170727200524</td>
<td>nginx-backup</td>
<td>Completed</td>
<td>0</td>
<td>0</td>
<td>2017-07-27 20:05:24 +0000 UTC</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

**NOTE:** The restore can take a few minutes to finish. During this time, the STATUS column reads InProgress.

After successful restoration, the STATUS column is Completed, and WARNINGS and ERRORS are 0. If there errors or warnings are obtained, you can read them in detail by running:

```bash
$ ark restore describe <RESTORE_NAME>
```

For more information, see [Debugging Information](#).

## BACKING UP MONARCH SWARM ENTERPRISE SERVER SERVICES

- **Data Engine Postgres**
  - Use the **kubectl** port-forwarding feature to access the database-server:
    ```bash
    $ kubectl port-forward -n <namespace> $(kubectl get pods -n <namespace> -l app=helm-release-depostgres -o jsonpath="{.items..metadata.name}") 5432:5432
    ```
  - Follow the official [PostgreSQL: Documentation: 9.6: Backup and Restore](#) documentation to proceed with backup and restoration (the server will be accessible in localhost:5432).

- **Core API Metadata Postgres**
  - Use **kubectl** port-forwarding feature to access database-server:
    ```bash
    $ kubectl port-forward -n <namespace> $(kubectl get pods -n <namespace> -l app=helm-release-postgres -o jsonpath="{.items..metadata.name}") 5433:5432
    ```
  - Follow the official [PostgreSQL: Documentation: 9.6: Backup and Restore](#) documentation to proceed with backup and restoration (the server will be accessible in localhost:5433).

- **Data Engine API Metadata Postgres**
  - Use **kubectl** port-forwarding feature to access database-server:
    ```bash
    $ kubectl port-forward -n <namespace> $(kubectl get pods -n <namespace> -l app=data-engine-api-postgres -o jsonpath="{.items..metadata.name}") 5434:5432
    ```
  - Follow official [PostgreSQL: Documentation: 9.6: Backup and Restore](#) to proceed backup-restore (the server will be accessible in localhost:5434).
- **Cassandra**

  - **Use the **kubectl**** port-forwarding feature to access the Cassandra server (only if JMX Security is configured for remote access, [JmxSecurity - Cassandra Wiki](#), – this not configured by default):

    ```
    $ kubectl port-forward -n <namespace> $(kubectl get pods -n <namespace> -l app=helm-release-cassandra -o jsonpath="{.items..metadata.name}")) 7199:7199 9042:9042
    ```

    (the server will be accessible in localhost:7199 - JMX, localhost:9042 - CQL)

  - **Use** **kubectl exec** **to execute any command in the pod (preferable). Set the tenant/namespace name in <namespace>:

    ```
    # '<namespace>-cassandra-0' - Cassandra pod name in Kubernetes dashboard
    # '<namespace>' - Kubernetes pod namespace
    # 'nodetool snapshot' - bash command, take a snapshot all keyspace
    $ kubectl exec -it <namespace>-cassandra-0 -n <namespace> nodetool snapshot
    ```

    After these commands are run, you can archive all of your snapshots and Cassandra data on the shared file system.

    Follow official [Backing Up and Restoring Data | Apache Cassandra 3.0](#) to proceed with backup/restore (the server will be accessible in localhost:7199 - JMX, localhost:9042 - CQL).

- **File-library**

  - **Use kubectl** to copy all data from the pod to some folder on your file system:

    ```
    $ export pod=$(kubectl get pods -n dev -l app=core-api -o jsonpath="{.items..metadata.name}"))
    $ kubectl cp <namespace>/${pod[0]}/var/swarm/file-library /backup/folder/
    ```