

Правообладатель: Общество с ограниченной ответственностью «Интернет для жизни»
(ООО «Интернет для жизни»)
430005, Республика Мордовия, г. Саранск, ул. Большевистская, д. 11, офис 201,
ОГРН 1081326002724, ИНН 1326207059

**ПЛАТФОРМА ОБРАБОТКИ ДАННЫХ
НА ОСНОВЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА**

Руководство по техническому обслуживанию

RU.86213171.61897-01 46

Листов 20

АННОТАЦИЯ

Настоящий документ содержит сведения, необходимые для эксплуатации программы «Платформа обработки данных на основе искусственного интеллекта», предназначенной для автоматизации процессов анализа и детектирования по фотограмметрическим данным объектов, с определением их типа, занимаемой площади и физического расположения.

Оформление программного документа «Руководство по техническому обслуживанию» произведено по требованиям ЕСПД (ГОСТ 19.101-77¹, ГОСТ 19.103-77², ГОСТ 19.104-78³, ГОСТ 19.105-78⁴, ГОСТ 19.508-79⁵).

¹ ГОСТ 19.101-77 ЕСПД Виды программ и программных документов

² ГОСТ 19.103-77 ЕСПД Обозначение программ и программных документов

³ ГОСТ 19.104-78 ЕСПД Основные надписи

⁴ ГОСТ 19.105-78 ЕСПД Общие требования к программным документам

⁵ ГОСТ 19.508-79 ЕСПД Руководство по техническому обслуживанию Требования к содержанию и оформлению

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ.....	4
2. ОБЩИЕ УКАЗАНИЯ.....	4
3. ТРЕБОВАНИЯ К ТЕХНИЧЕСКИМ СРЕДСТВАМ	5
4. ОПИСАНИЕ ФУНКЦИЙ	5

1. ВВЕДЕНИЕ

Данный документ представляет собой руководство по техническому обслуживанию программы «Платформа обработки данных на основе искусственного интеллекта» (далее – Платформа).

Документ предназначен специалисту по техническому обслуживанию и администратору Платформы на этапе сопровождения.

При осуществлении работ также следует руководствоваться следующими документами:

- RU.86213171.61897-01 13 Описание программы
- RU.86213171.61897-01 31 Описание применения
- RU.86213171.61897-01 32 Руководство системного программиста
- RU.86213171.61897-01 33 Руководство программиста
- RU.86213171.61897-01 34 Руководство оператора

2. ОБЩИЕ УКАЗАНИЯ

2.1. Порядок технического обслуживания

В рамках сопровождения Платформы специалистом по техническому обслуживанию реализуется выполнение следующих функций:

- Обновление программного обеспечения;
- Выполнение резервного копирования;
- Осуществление проверки штатными средствами работоспособности

Платформы;

- Восстановление работоспособности Платформы при аварийных ситуациях.

Требования к уровню подготовки специалиста по техническому обслуживанию приведены в п.2.2. настоящего программного документа.

2.2. Требования к персоналу

Выполнение функций, указанных в п.2.1. данного программного документа, предполагает соответствие специалиста по техническому обслуживанию следующим требованиям:

- Системное администрирование Linux;
- Администрирование СУБД MongoDB, PostgreSQL;
- Администрирование систем контейнеризации Docker, кластеризации Docker Swarm;
- Навыки работы с системой мониторинга Prometheus и системы визуализации Grafana.

3. ТРЕБОВАНИЯ К ТЕХНИЧЕСКИМ СРЕДСТВАМ

3.1. Минимальный состав технических средств программы

Рекомендуемые требования к аппаратной части, на которой планируется развёртывание и функционирование Платформы:

- Процессор: 16x CPU 3.5 GHz.
- Оперативная память: 64 Gb.
- Видео карта: NVIDIA GeForce RTX 3080, 10 Gb.
- SSD накопитель: 1 Tb SSD.
- Жесткий диск: 18 Tb HDD.
- Сетевой адаптер: Gigabit Ethernet 1000 Mbit.

4. ОПИСАНИЕ ФУНКЦИЙ

4.1. Обновление программного обеспечения

В связи с тем, что Платформа построена на микросервисной

архитектуре, где в качестве среды контейнеризации используется Docker и для отказоустойчивости серверы объединены в кластер Docker Swarm, то под обновлением программного обеспечения Платформы понимается обновление образов Docker (обновление образом Docker программных микросервисов Платформы), из которых разворачиваются контейнеры.

Для обновления микросервиса новым образом нужно:

1. Подключиться к любому серверу кластера Docker Swarm по протоколу `ssh` от имени пользователя, обладающего правами суперпользователя:

`ssh someuser@server-ip-address` – из среды Linux,

либо с помощью программы Putty из среды Windows.

2. Выполнить поиск микросервиса, который нужно обновить, по его имени. Например:

`sudo docker service ls |grep kpp`

Примерный вывод команды:

```
x5js3e9e4j9j      aip_sier-kppreplicated  1/1
newharbor.evolenta.ru/sier/kpp:v45-mysql *:81->80/tcp, *:9192->9192/tcp
```

3. Обновить образ, используя имя микросервиса, найденное в п.2.

Например, для обновления микросервиса `aip_sier-kpp` нужно выполнить:

`docker service update --with-registry-auth --image`

`newharbor.evolenta.ru/sier/kpp:v46-mysql aip_sier-kpp,`

где ключ (`image`) – позволяет ввести имя нового образа.

4. Проверить состояние микросервиса командой:

`sudo docker service ls |grep aip_sier-kpp`

Примерный вывод команды:

```
x5js3e9e4j9j      aip_sier-kpp      replicated  1/1
newharbor.evolenta.ru/sier/kpp:v46-mysql *:81->80/tcp, *:9192->9192/tcp
```

Убедиться, что количество запущенных и запланированных к запуску реплик микросервиса одинаково (`replicated 1/1`).

Если управление микросервисами осуществляется с помощью графической оболочки программного обеспечения «Portainer» (устанавливается отдельно), то для обновления необходимо:

1. Авторизоваться в программном обеспечении «Portainer» с правами администратора по адресу:

http://ip-address:9000,

где ip-address – ip-адрес или доменное имя, на котором развернуты микросервисы.

2. Открыть пункт меню «Services» и выбираем микросервис, который требуется обновить, нажав на имя левой кнопкой мыши. При необходимости осуществить поиск микросервиса по его наименованию, заполнив поле «Search» (Рисунок 1).

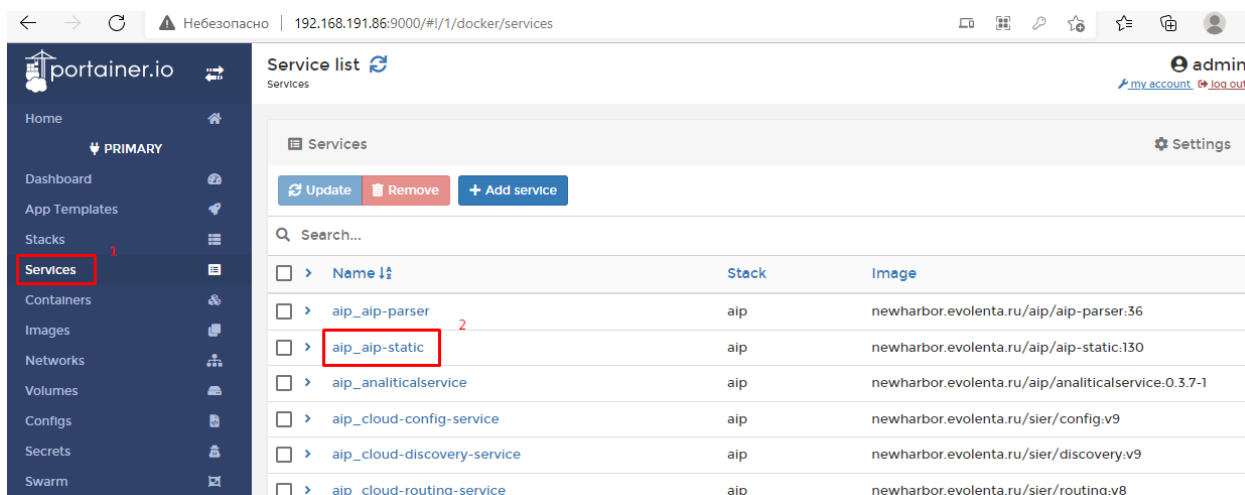


Рисунок 1 – Интерфейс пункта меню «Services» в программном обеспечении Portainer

3. Выбрать пункт меню «Container image» (Рисунок 2).

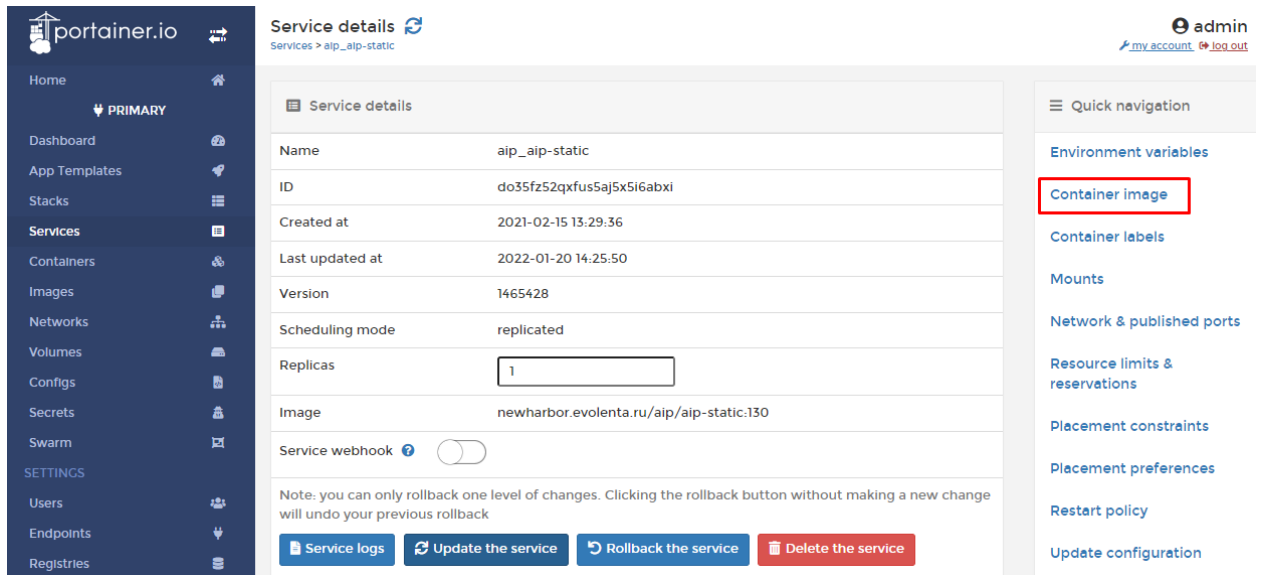


Рисунок 2 – Пункт меню «Container image» в программном обеспечении Portainer

4. В режиме «Simple mode» в поле «Image» указать новое имя образа – например, newharbor.evolenta.ru/sier/kpp:v46-mysql (Рисунок 3).

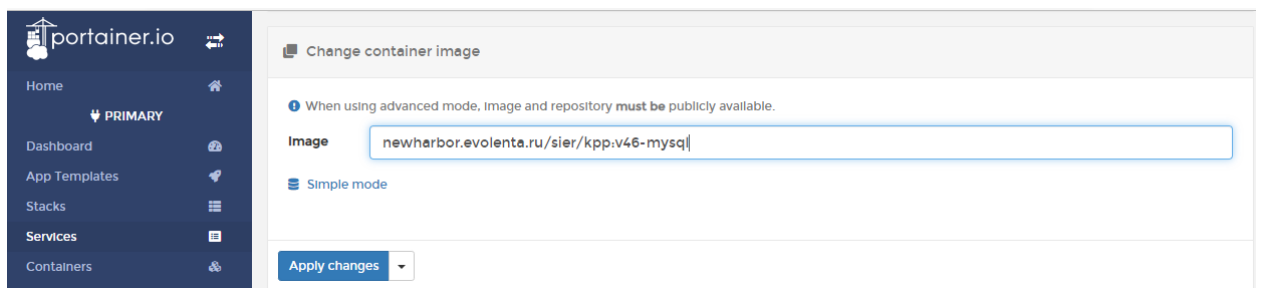


Рисунок 3 – Интерфейс пункта меню «Container image» в программном обеспечении Portainer

5. Нажать «Apply changes» в интерфейсе пункта меню «Container image» в программном обеспечении Portainer.

6. Перейти в пункт меню «Services» и проверить, что количество запущенных контейнеров соответствует требуемому (Рисунок 4).

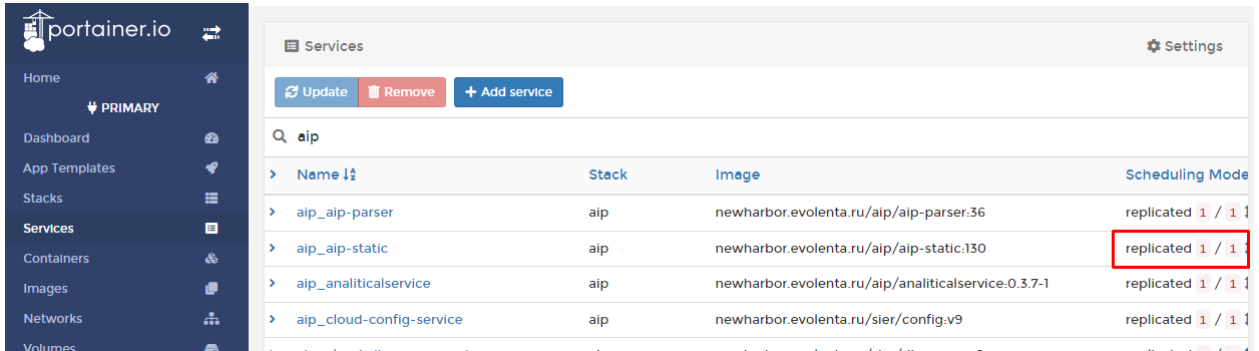


Рисунок 4 – Интерфейс пункта меню «Container image» в программном обеспечении Portainer

4.2. Выполнение резервного копирования

Поскольку все микросервисы Платформы являются сервисами без сохранения состояния (stateless), то в их резервном копировании нет необходимости. Однако микросервисы Платформы настроены на работу с внешними базами данных (БД), такими как MySQL, MongoDB, PostgreSQL, которые нуждаются в резервировании.

Для резервного копирования БД используются штатные средства упомянутых СУБД, а именно команды: *mysqldump*, *mongodump*, *pg_dump*, соответственно. Ниже представлены примеры использования каждой команды:

1. Резервирование БД модуля авторизации и аутентификации КПП с последующей архивацией осуществляется следующей командой:

```
mysqldump -h ${MYSQL_HOST} -P ${MYSQL_PORT} -u ${MYSQL_USER}
${DATABASE_NAME} | gzip > archive_name.sql.gz,
```

где:

-h и -P – ip-адрес сервера и порт, на которых запущена СУБД MySQL,

-u – имя пользователя,

DATABASE_NAME – имя резервируемой БД.

2. Для резервирования БД системы аналитики Pentaho, БД платформы моделирования и создания бизнес-процессов Camunda, и БД геоданных, с

последующей архивацией используется следующая команда:

```
pg_dump -h ${PGSQL_HOST} -p ${PGSQL_PORT} -U ${PGSQL_USER}
${DATABASE_NAME} | gzip > archive_name.sql.gz,
```

где:

-h и -p – ip-адрес сервера и порт, на которых запущена СУБД PostgreSQL,

-U – имя пользователя,

DATABASE_NAME – имя резервируемой БД.

3. Резервное копирование БД системы обработки фотограмметрических данных (СОФД) с последующей архивацией выполняется следующей командой:

```
mongodump -h ${MONGO_HOST} --port ${MONGO_PORT} -u
${MONGO_USER} --authenticationDatabase ${AUTH_DATABASE_NAME} --db
${DATABASE_NAME} --gzip --archive= archive_name.gz,
```

где:

-h и --port – ip-адрес сервера и порт, на которых запущена СУБД MongoDB,

-u – имя пользователя,

--authenticationDatabase – БД, в которой авторизуется пользователь,

DATABASE_NAME – имя резервируемой БД.

Данные команды резервирования могут запускаться как вручную из командной строки, так и по расписанию с помощью команды crontab. Например, чтобы резервирование осуществлялось ежедневно в 01:00, нужно:

1) открыть планировщик crontab в режиме редактирования, путем ввода следующей команды:

```
sudo crontab -e
```

2) нажать клавишу I на клавиатуре и поместить туда следующую строку:

```
00 01 * * * /путь/до/файла/mongo_backup.sh >/dev/null,
```

где:

mongo_backup.sh – bash-скрипт с командой для резервирования БД Mongo.

3) нажать на клавиатуре

:wq,

чтобы сохранить изменения и выйти.

Проверить внесенные в планировщик изменения можно командой:

sudo crontab -l.

4.3. Осуществление проверки работоспособности Платформы

В качестве средств диагностики работы Платформы могут использоваться команды ОС Linux, а также сторонняя система мониторинга, построенная на системе сбора метрик «Prometheus» и системе визуализации «Grafana».

Проверка работоспособности Платформы с помощью командами ОС Linux осуществляется поочередным подключением к серверам, составляющим кластер Docker Swarm, по протоколу ssh, как показано в пункте 4.1.1, и выполнением команд:

1. *top*

На экран монитора будет выведено, кроме данных о доступной и используемой оперативной памяти (ОЗУ), файле подкачки и процессорах (для этого нужно нажать цифру 1 на клавиатуре), список процессов, наиболее сильно нагружающих Платформу. Пример результата использования команды ОС Linux *top* представлен на Рисунке 5.

```

top - 15:36:40 up 49 days, 6:24, 1 user, load average: 2.99, 2.99, 2.28
Tasks: 917 total, 3 running, 911 sleeping, 0 stopped, 3 zombie
%Cpu0  :  0.0 us,  0.4 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,  1.9 si,  0.0 st
%Cpu1  :  0.8 us,  1.5 sy,  0.0 ni, 97.3 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
%Cpu2  :  0.8 us,  1.9 sy,  0.0 ni, 97.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.4 us,  0.0 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  1.1 us,  0.0 sy,  0.0 ni, 98.5 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
%Cpu5  :  0.8 us,  0.0 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 68.2 us,  0.0 sy,  0.0 ni, 31.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.4 us,  0.8 sy,  0.0 ni, 98.5 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
%Cpu8  :  0.4 us,  0.0 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.8 us,  1.2 sy,  0.0 ni, 98.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  2.3 us,  2.3 sy,  0.0 ni, 95.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  2.3 us,  3.1 sy,  0.0 ni, 94.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.4 us,  0.4 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.4 us,  0.0 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.8 us,  0.0 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  1.1 us,  0.4 sy,  0.0 ni, 98.1 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
%Cpu16 :  0.8 us,  0.0 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  2.3 us,  1.1 sy,  0.0 ni, 96.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 : 31.8 us,  0.0 sy,  0.0 ni, 68.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.8 us,  0.4 sy,  0.0 ni, 98.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  1.5 us,  1.2 sy,  0.0 ni, 97.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  1.5 us,  1.9 sy,  0.0 ni, 96.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  1.1 us,  1.5 sy,  0.0 ni, 96.9 id,  0.4 wa,  0.0 hi,  0.0 si,  0.0 st
Mib Mem : 64300.4 total, 6124.0 free, 46559.8 used, 11616.5 buff/cache
Mib Swap : 40960.0 total, 23227.8 free, 17732.2 used, 16296.4 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 8480 root        20   0   4276    544    520  R  99.6   0.0  70911:54  sh
3553140 root        20   0   946952 213832 19816  S  13.4   0.3   16:23.80  cadvisor
 1358 root        20   0   6478648 276772 24544  S   3.1   0.4   567:01.96  dockerd
19712 root        20   0    24.0g   2.3g 12860  S   3.1   3.7   1192:41   java
 8886 root        20   0   667976   20756 1832  S   1.5   0.0   918:21.32  python
11723 root        20   0   616960   90304   4  S   1.5   0.1   957:56.62  python
2207984 root        20   0 1127020 48756   5152  S   1.5   0.1   16:45.86  python
1200287 root        20   0 1212032 49360   4392  S   1.1   0.1   117:01.14  python
1220217 root        20   0   544236   7248  2476  S   1.1   0.0   115:49.05  python
2330555 root        20   0 1159092 220512 34420  S   1.1   0.3   630:06.28  python3
 10426 root        20   0    19.1g   6.2g 11880  S   0.8   9.9   813:04.79  java
 12199 root        20   0 115968    4764   0  S   0.8   0.0   496:52.57  python
16486 root        20   0 117876    5200   92  S   0.8   0.0   519:37.72  python
105216 root        20   0 1070888 102356 3700  S   0.8   0.2   245:50.29  python3
3586806 systemd+  20   0 216108   13280 11232  S   0.8   0.0   0:01.57   postgres
  11 root        20   0   0   0   0  I   0.4   0.0   35:29.35   rcu_sched
 1301 mongod     20   0    18.6g   16.5g 12588  S   0.4  26.3   207:21.57  mongod
 8304 root        20   0 6110492 85780   5944  S   0.4   0.1   71:06.06   java
12676 root        20   0 8262920 55560   1880  S   0.4   0.1   52:13.18   java
16103 root        20   0 9594568 147132  4236  S   0.4   0.2   62:28.92   java
19759 systemd+  20   0 2235020 308876   0  S   0.4   0.5   243:00.37  mongod
595319 user        20   0    13.3g   9.1g  3540  S   0.4  14.4   348:01.04  python
600906 user        20   0 175416   7844  2044  S   0.4   0.0   34:59.79  python
1349141 user        20   0 175168   7768  2220  S   0.4   0.0   34:57.20  python
1373302 user        20   0 225216   98440 2364  S   0.4   0.1   36:46.51  python
1379206 user        20   0 179452   11892 2432  S   0.4   0.0   35:26.05  python
3276751 root        20   0 1176464 18924  5216  S   0.4   0.0   0:13.97   unicorn
3580851 user        20   0    8976   4768  3304  R   0.4   0.0   0:05.28   top
3589804 root        20   0   0   0   0  I   0.4   0.0   0:00.01   kworker/10:1-events
3589869 root        20   0   0   0   0  I   0.4   0.0   0:00.01   kworker/4:1-events
3618909 mysql      20   0 4583856 14648  6448  S   0.4   0.0   5:44.00   mysqld
  1 root        20   0 174176   10824 6416  S   0.0   0.0   37:13.31  systemd
  2 root        20   0   0   0   0  S   0.0   0.0   0:04.74   kthreadd
  3 root        0 -20   0   0   0  I   0.0   0.0   0:00.00   rcu_gp
  4 root        0 -20   0   0   0  I   0.0   0.0   0:00.07   rcu_par_gp
  6 root        0 -20   0   0   0  I   0.0   0.0   0:00.00   kworker/0:0H-kblockd
  9 root        0 -20   0   0   0  I   0.0   0.0   0:00.00   mm_percpu_wq
 10 root        20   0   0   0   0  S   0.0   0.0   1:06.53   ksoftirqd/0
 12 root        rt   0   0   0   0  S   0.0   0.0   0:06.55   migration/0
  
```

Рисунок 5 – Пример результата использования команды ОС Linux *top*

Следует обратить внимание на размер используемой оперативной памяти (Mib Mem / %Mem) и файла подкачки (Mib Swap / %Swap) – значения не должны быть выше 98% для обеспечения работоспособности Платформы.

Активное использование операционной системой (ОС) файла подкачки говорит о нехватки ОЗУ на вычислительных мощностях.

Также следует принять во внимание значение средней нагрузки (Load Average). Оптимальным считается значение, равное количеству ядер в ОС.

При значительном превышении указанных величин следует проанализировать лог-файлы ОС и приложений, как правило располагающиеся в директории `/var/log`, и, при необходимости, выполнить оптимизацию используемого на сервере ПО либо добавить ОЗУ и увеличить мощность процессоров.

2. `sudo docker container stats`

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
1c794870421e	alpdev_alp-static.1.c6ky1p4y75jom17uJ0efzoxu	0.00%	10.01MiB / 62.79GiB	0.02%	470KB / 3.39MB	1.35MB / 0B	82
461502e80045	monitoring_grafana.1.6gh170zdf6k7boottg8j4d4bc	0.04%	47.23MiB / 62.79GiB	0.07%	921KB / 16.7MB	2.15MB / 4.93MB	29
4694007c4ec3	monitoring_blackbox-exporter.1.u9h2mamlun2etrdk48vj4xuj	0.01%	20.14MiB / 62.79GiB	0.03%	8.83MB / 3.66MB	8.19KB / 0B	25
20f88f1636d3	monitoring_cadvisor.1.zr9b181wad13x25156f1dr1yh	9.49%	217.3MiB / 62.79GiB	0.34%	7.61MB / 573MB	4.1KB / 0B	32
1562bd1d4300	monitoring_node-exporter.1.obe8dct4ky9vgzcd41n17px	0.00%	21.26MiB / 62.79GiB	0.03%	470KB / 11.7MB	0B / 0B	50
F2d2954e305	monitoring_alertmanager.1.z4bnu8vk7w2v101m064rml6rr	0.03%	18.99MiB / 62.79GiB	0.03%	172KB / 125KB	279KB / 36.9KB	30
0d0030a34093	monitoring_prometheus.1.sq2qepk7x2vq13x1duxqztzd	0.00%	477.4MiB / 62.79GiB	0.74%	629MB / 26.6MB	336KB / 13.1MB	29
006a9b330f3	geostorages_dev_mapproxy.1.1z6hd7zfoezm8i13a9rtj7a	0.00%	75.31MiB / 62.79GiB	0.12%	4.77MB / 14.6MB	39.1MB / 24.6KB	22
03e670aa7992	geostorages_dev_admler.1.mdf4c70f3pm1jzph3j5mp	0.00%	6.262MiB / 62.79GiB	0.01%	6.57KB / 0B	67.1MB / 0B	1
2f0fa7d372a2	geostorages_dev_db.1.vfcpkrnxhdeakoy6yztzxs46	0.00%	3.536GiB / 62.79GiB	5.63%	10.2MB / 22.6MB	6.63GB / 172MB	13
541b0217d513	geostorages_dev_geoserver-api-layer.1.v381gaaoysd1h6x7qdtjg8o2p	0.01%	128.3MiB / 62.79GiB	0.20%	10.3MB / 8.99MB	31MB / 164MB	3
80fa2a38a842	geostorages_dev_redis.1.h027r2xgzj20iatw6zmlagv	0.00%	4.211MiB / 62.79GiB	0.01%	3.21KB / 0B	2.67MB / 0B	5
c3a7b480c204	pdps-processing_dev_masks_binartization.1.kybbat8q97l041kxj3j6awn1	0.00%	58MiB / 62.79GiB	0.09%	6.29MB / 7.32MB	9.38MB / 0B	2
a43d38900ee9	alpdev_pkpprovider.1.j7f761t7rvh545sadszmzyoe	0.99%	91.59MiB / 62.79GiB	0.14%	16.7KB / 0B	4.01MB / 5.67MB	4
56f2b24f7322	pdps-processing_dev_directory-cleaner.1.x1fzdyk5nrmvy48qt4v6w88j	0.00%	33.09MiB / 62.79GiB	0.05%	21.2KB / 3.77KB	108MB / 0B	3
33e19b06da86	alpdev_analitica1service.1.kpzb9rbj3f7ab512kbv9x6h9	1.01%	72.64MiB / 62.79GiB	0.11%	22.4MB / 48.4MB	37.9MB / 0B	5
ea2c0665182c	alpdev_sier-kpp.1.4cq9xsnq804mxm90k1679vr	0.03%	159.9MiB / 62.79GiB	0.25%	4.8GB / 4.67GB	332MB / 4.1KB	84
10af2b4ed64e	geostorages_dev_pkp_proxy.1.rfyesu4ajrawrp3b445bfh5vg	1.21%	12.23MiB / 62.79GiB	0.02%	13.1MB / 11.8MB	680KB / 0B	4
0889286ae44	pdps-processing_dev_cpis_executor.1.zna6p2eak4pf1v0k174ovon2	0.00%	83.9MiB / 62.79GiB	0.13%	616MB / 719MB	2.11GB / 324KB	6

Рисунок 6 – Пример результата использования команды ОС Linux `sudo docker container stats`

Данная команда является полезной, если надо узнать сколько каждый из запущенных контейнеров Docker использует ОЗУ (Mem %) и процессорного времени (CPU %), а также сетевую активность (Net I/O) и активность на уровне блочного устройства (Block I/O).

Также для мониторинга работоспособности Платформы используется связка из сторонних систем «Prometheus» и «Grafana». Система сбора метрик «Prometheus» собирает разного рода метрики с серверов кластера Docker Swarm и запущенных на них контейнерах, на основе которых система визуализации «Grafana» строит графики.

Для доступа в графической оболочке системы «Grafana» нужно перейти по ссылке:

http://server-ip-address:3000,

ГДЕ:

server-ip-address – ip-адрес или доменное имя сервера, на котором развернута система «Grafana».

Система «Grafana» отображает графики, касающиеся работы серверов. Пример визуализации графиков в системе «Grafana» представлен на рисунке 7.

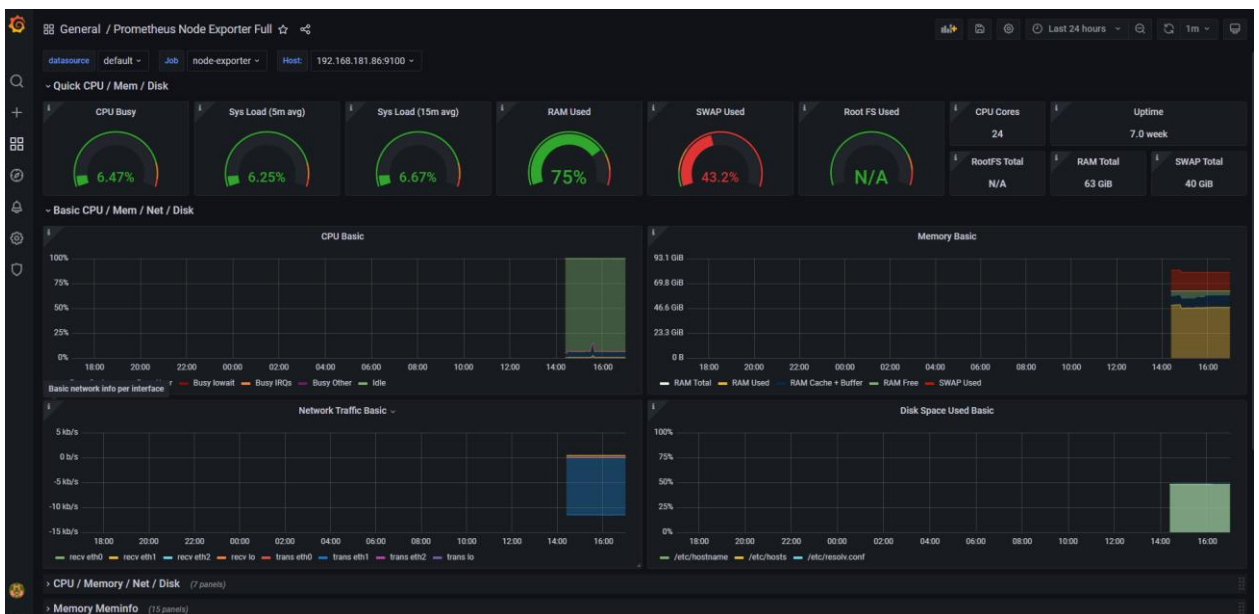


Рисунок 7 – Пример визуализации графиков в системе «Grafana»

Кроме того, система «Grafana» визуализирует детальную информацию, получаемую от контейнеров (представлено на рисунке 8).

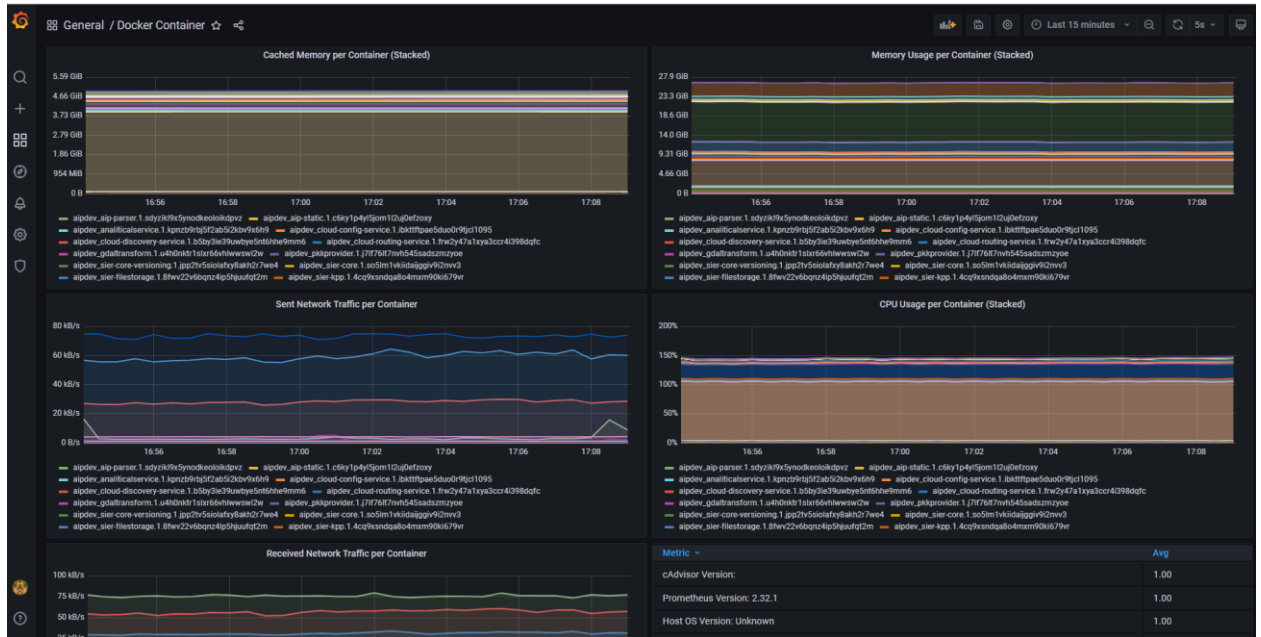


Рисунок 8 – Пример визуализации детальной информации, получаемой от контейнеров, в системе «Grafana»

Среди выводимых параметров (детальной информации, получаемой от контейнеров) в системе «Grafana» – использование ОЗУ, процессорного времени, средняя нагрузка, работа дисковой подсистемы, сетевой трафик и другие.

Основываясь на полученной информации, администратор в рамках технического обслуживания может увидеть проблемы в работе Платформы, а также спрогнозировать возможное появление проблем.

Указанная система мониторинга также способна отправлять оповещения на электронную почту или в мессенджеры при достижении контролируемыми параметрами пороговых значений.


4.4. Восстановление работоспособности Платформы при аварийных ситуациях

Восстановление работоспособности Платформы требует выявления причин возникновения аварийной ситуации. Для этого нужно проделать следующие действия:

1. Убедиться в работоспособности всех серверов в кластере Docker Swarm. Для этого необходимо подключиться к любому серверу кластера по протоколу ssh, как описано в пункте 4.1.1. настоящего документа, и выполнить команду:

```
sudo docker node ls
```

Пример результата использования команды ОС Linux *sudo docker node ls* представлен на рисунке 9.

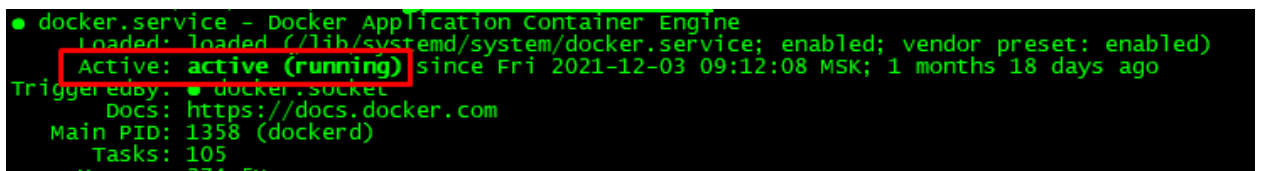


```
[efolenta@mfc-app-arb ~]$ sudo docker node ls
ID                HOSTNAME          STATUS      AVAILABILITY  MANAGER STATUS  ENGINE VERSION
0uu7q3hdrd71485ndyxq461 *  mfc-app-arb     Ready       Active         Reachable       18.03.1-ce
kv279pfgp4cj025befm3dc1gu  mfc-app-rep1    Ready       Active         Leader          18.03.1-ce
cx15w3r2nvtrom1ldavhvn3q2  mfc-app-rep2    Ready       Active         Reachable       18.03.1-ce
```

Рисунок 9 – Пример результата использования команды ОС Linux *sudo docker node ls*

Следует внимательно ознакомиться с результатами использования команды ОС Linux *sudo docker node ls*, а именно обратить внимание на значения полей «Status» и «Availability» – должны быть значения «Ready» и «Active», соответственно. Если какое-то из значений отличается от указанных, то необходимо подключиться к соответствующему серверу по протоколу ssh и проверить состояние процесса docker путем ввода следующей команды:

```
sudo systemctl status docker
```



```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2021-12-03 09:12:08 MSK; 1 months 18 days ago
     TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
   Main PID: 1358 (dockerd)
   Tasks: 105
   Memory: 774.5M
```

Рисунок 10 – Пример результата использования команды ОС Linux *sudo systemctl status docker*

Если в результатах использования команды ОС Linux *sudo systemctl status docker* статус отличается от «Active (Running)», то следует запустить службу *systemctl* путем ввода следующей команды:

sudo systemctl start docker

либо перезапустить данную службу в случае наличия ошибок путем ввода следующей команды:

sudo systemctl restart docker

2. Проверить состояние микросервисов. Для этого необходимо убедиться, что, в соответствии с пунктом 4.1.4 или 4.1.9, количество запущенных и запланированных к запуску реплик микросервиса одинаково (*replicated 1/1*). Если количество запущенных контейнеров равно нулю, то следует проанализировать лог-файлы микросервиса на наличие ошибок. Для этого необходимо подключиться к любому серверу кластера Docker Swarm по протоколу ssh, как описано в пункте 4.1.1, и выполнить команду:

sudo docker service logs service-name,

где

service-name – имя микросервиса, лог-файлы которого следует проанализировать.

3. Для просмотра лог-фалов из графической оболочки «Portainer» необходимо выполнить пункты 4.1.5 и 4.1.6. Затем нажать левой кнопки мыши на кнопку «>» для раскрытия списка контейнеров, и кнопку «Logs» в столбце «Actions». Интерфейс графической оболочки «Portainer», с указанием графических элементов для просмотра лог-фалов, представлен на рисунке 11.

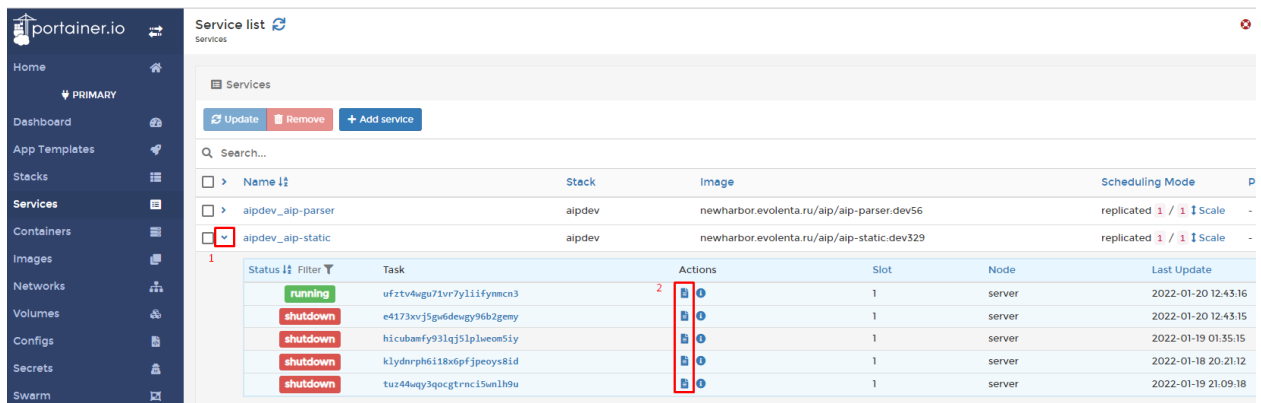


Рисунок 11 – Интерфейс графической оболочки «Portainer», с указанием графических элементов для просмотра лог-фалов

После совершения вышеописанных действия на экран монитора будет выведено содержимое лог-файла проблемного микросервиса, где по ключевому слову «Error» можно выявить причину неработоспособности. После устранения причины, контейнер будет запущен автоматически.

4. В случае, если контейнер микросервиса запущен, но анализ лог-файлов показал, что процесс внутри контейнера не функционирует должным образом, то следует осуществить перезапуск контейнера. Для этого из командной строки Linux нужно определить идентификатор (id) проблемного контейнера путем ввода следующей команды:

```
sudo docker container ls |grep aip-static
```

Пример результата использования команды ОС Linux *sudo docker container ls |grep aip-static*:

```
36a787413a73    newharbor.evolenta.ru/aip/aip-static:dev329    "httpd-  
foreground"    23 hours ago    Up 23 hours    80/tcp  
aipdev_aip-static.1.ufztv4wgu71vr7yliifynmcn3
```

Затем использовать этот id для перезапуска контейнера путем ввода следующей команды:

```
sudo docker container kill 36a787413a73
```

и, в соответствии с пунктом 4.1.7, проверить, что количество запущенных контейнеров соответствует требуемому и что в лог-файлах отсутствуют ошибки.

Для выполнения перезапуска контейнера из графической оболочки «Portainer» необходимо выполнить пункты 4.1.5 и 4.1.6. Затем выбрать пункт меню «Containers», выделить проблемный контейнер, поставив галочку слева от его имени, и нажать на кнопку «Kill». Интерфейс графической оболочки «Portainer», с указанием графических элементов для выполнения перезапуска контейнера, представлен на рисунке 12.

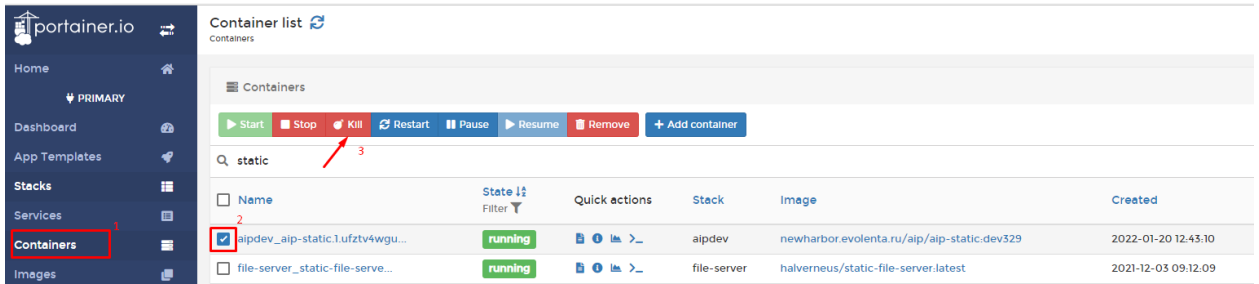


Рисунок 12 – Интерфейс графической оболочки «Portainer», с указанием графических элементов для выполнения перезапуска контейнера

После перезапуска контейнера из графической оболочки «Portainer» необходимо проверить сообщения в лог-файлах на наличие ошибок в соответствии с пунктом 4.4.3.

