«СИСТЕМА ИСПОЛНЕНИЯ ЭЛЕКТРОННЫХ РЕГЛАМЕНТОВ И МЕЖВЕДОМСТВЕННОГО ВЗАИМОДЕЙСТВИЯ»

РУКОВОДСТВО АДМИНИСТРАТОРА

(том 2 - Инструкция по развертыванию платформы)

на 32 листах

2022

Содержание

Перечень терминов и сокращений

Термин, сокращение	Определение
ID	Идентификатор – уникальный признак объекта, позволяющий отличать его от других объектов
СИЭР, Система, Платформа	«Система исполнения электронных регламентов и межведомственного взаимодействия»
БП	Бизнес-процесс
API	(Application Programming Interface — программный интерфейс приложения) набор способов и правил, по которым различные программы общаются между собой и обмениваются данными
СМЭВ	Система межведомственного электронного взаимодействия
СУБД	Система управления базами данных
БД	База данных

1 Введение

Полное наименование системы: «Система исполнения электронных регламентов и межведомственного взаимодействия».

Условное наименование: СИЭР, Система, Платформа.

Настоящий документ представляет собой руководство администратора программного обеспечения Системы для развертывания платформы.

1.1 Назначение и условия применения

«Система исполнения электронных регламентов и межведомственного взаимодействия» предназначена для формирования единого информационного пространства для всех пользователей Системы – сотрудников органов исполнительной власти субъекта Российской Федерации, уполномоченных на осуществление государственного контроля (надзора) на территории соответствующих субъектов Российской Федерации, органов местного самоуправления, уполномоченных в соответствии с федеральными законами на осуществление муниципального района.

1.2 Область применения

Целью функционирования Платформы является предоставление участникам контрольной (надзорной) деятельности высокотехнологичного интеллектуального инструмента с полным набором цифровых сервисов, которые позволят:

- обеспечить прозрачность деятельности ведомств;
- создать среду доверия для граждан и организаций;
- провести цифровую трансформацию государственных и муниципальных органов контроля (надзора) и перейти на качественно новый уровень проведения надзорных мероприятий, основанный на учете только тех требований, нарушение которых может привести к ущербу;

- принимать решения на основе объективной, оперативной и регулярно собираемой информации;
- повысить эффективность и результативность деятельность контрольных (надзорных) органов за счет оперативного выявления признаков нарушения по результатам анализа рисков, основанного на массивах «больших данных»;
- предотвращать нарушения за счет применения методов предиктивной аналитики и возможности проведения индивидуализированных профилактических мероприятий;
- формировать отчетность и аналитические материалы по результатам осуществления контроля (надзора):
- осуществлять интеграционное взаимодействие со СМЭВ:
  - отправка запроса;
  - чтение ответов;
  - просмотр результатов.

2 Инструкция по развертыванию Системы для автоматизации бизнес-процессов

2.1 Обзор системы

В Системе используется концепция микросервисов. Концепция микросервисов предполагает, что весь функционал приложения поделен на части, и каждая часть выделена в отдельный сервис. Благодаря этому появляется возможность «горизонтального» масштабирования системы и динамического обновления компонентов без остановки работы всего приложения.

Для упрощения управления десятками микросервисов в архитектуру системы введен сервис конфигурирования, который предназначен для централизованного хранения и раздачи конфигурационных файлов всем остальным сервисам системы.

Непосредственно приложение СИЭР состоит из следующих компонентов и сервисов:

– **«статика»**, клиентские компоненты для отображения в браузере пользователей;

– **сервис ядра системы**, реализует обработку основных вызовов пользователей для работы с сущностями (поиск/создание/обновление/удаление сущностей);

– **сервис бизнес-логики**, реализует API основной логики работы учетной системы;

– **сервис шлюза файлохранилища**, реализует API для работы с файлами, загружаемыми или создаваемыми при работе системы (загрузка файлов в хранилище, получение их из хранилища, удаление);

– **сервис нумерации**, предоставляет другим сервисам возможность получать уникальные автоинкрементные идентификаторы для своих внутренних нужд. Является синглетом и не масштабируется горизонтально (запрещается поднимать более одного инстанса этого сервиса, чтобы не нарушить автоинкрементные последовательности);

– **сервис рендеринга форм**, взаимодействует с рендером, предоставляет API для генерации печатных форм другим микросервисам;

– **рендер форм**, реализует логику рендеринга печатных форм;

– **СМЭВ-клиент**, предоставляет API для отправки межведомственных запросов, реализует логику взаимодействия с внешними интеграционными компонентами (шина интеграции, ESB) и удаленными сервисами.

2.2 Установка и настройка MongoDB

2.2.1 Установка MongoDB 4.2. на RHEL 8.

2.2.1.1 Создаем файл .repo для репозитория монги:

sudo vi /etc/yum.repos.d/mongodb-org.repo

2.2.1.2 Заполняем его следующим содержимым:

[mongodb-org-4.2]

name=MongoDB Repository

baseurl=https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.2/x86_64/

gpgcheck=1

enabled=1

gpgkey=https://www.mongodb.org/static/pgp/server-4.2.asc

2.2.1.3 Проверяем, что репозиторий доступен:

yum repolist

В списке репозиториев должен фигурировать репозиторий MongoDB Repository.

2.2.1.4 Устанавливаем MongoDb:

sudo yum install mongodb-org

2.2.1.5 Добавляем в автозапуск:

sudo systemctl enable mongod

2.2.1.6 Запускаем:

sudo systemctl start mongod

2.2.2. Первичная инициализация (создание базы данных и включение аутентификации).

Сразу после установки MongoDB сконфигурирована на работу по localhost с рутовым подключением без какой-либо авторизации (не запрашивается даже имя пользователя). Необходимо провести реконфигурацию СУБД с включением аутентификации и разрешением сетевого взаимодействия.

2.2.2.1 Подключаемся к СУБД Mongo консольным клиентом:

```
mongo
```

Вывод консоли при успешном подключении (начальная часть):

```
MongoDB shell version v 4.2.17
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.2.17
Welcome to the MongoDB shell.
…
```

2.2.2.2 Переключаемся на служебную базу данных:

```
use admin
```

Вывод консоли при успешной операции:

```
switched to db admin
```

2.2.2.3 Создаем пользователя-администратора «root»:

```
db.createUser(
  {
    user: "root",
    pwd: "pass",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Вывод консоли при успешном создании:

```
Successfully added user: {
        "user" : "root",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

2.2.2.4 Проверяем, что пользователь создан:

```
show users
```

Вывод консоли при успешной проверке:

```
{
        "_id" : "admin.root",
        "user" : "root",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

2.2.2.5 Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.2.2.6 Останавливаем демон для дальнейшего включения авторизации:

systemctl stop mongod

2.2.2.7 Открываем конфиг MongoDB для редактирования:

vi /etc/mongod.conf

2.2.2.8 Изменяем секцию # network interfaces :

меняем bindIp: 127.0.0.1 на bindIp: 0.0.0.0 для включения возможности подключения с других серверов

2.2.2.9 Раскомментируем секцию #security и добавляем туда блок включения авторизации:

```
security:
  authorization: enabled
```

2.2.2.10 Запускаем демон MongoDB:

systemctl start mongod

2.2.2.11 Подключаемся консольным клиентом:

mongo

Вывод консоли при успешном подключении (начальная часть):

MongoDB shell version v 4.2.17
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.2.17

2.2.2.12 Переключаемся на служебную базу данных:

use admin

Вывод консоли при успешной операции:

switched to db admin

2.2.2.13 Авторизуемся рутом:

db.auth("root","pass")

Вывод консоли при успешной авторизации:

1

2.2.2.14 Создаем БД приложения:

use knd

Вывод консоли при успешном переключении на БД sier:

switched to db knd

2.2.2.15 Создаем пользователя для подключения к базе данных:

```
db.createUser(
  {
    user: "knd",
    pwd: "pass",
    roles: [ {role: "readWrite", db: "knd"} ]
  }
)
```

Вывод консоли при успешном добавлении пользователя:

```
Successfully added user: {
        "user" : "knd",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "knd"
                }
        ]
}
```

2.2.2.16 Создаем БД рендера:

use renderdb

Вывод консоли при успешном переключении на БД renderdb:

```
switched to db renderdb
```

2.2.2.17 Создаем пользователя для подключения к базе данных:

```
db.createUser(
  {
    user: "renderdb",
    pwd: "pass",
    roles: [ {role: "readWrite", db: "renderdb"} ]
  }
)
```

Вывод консоли при успешном добавлении пользователя:

```
Successfully added user: {
        "user" : "renderdb",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "renderdb"
                }
        ]
}
```

2.2.2.18 Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.2.3. Конфигурирование MongoDb replica set для распределенных отказоустойчивых конфигураций на RHEL 8.

Перед настройкой Replica Set на все серверы нужно установить и настроить MongoDB, согласно п.п.2.2-2.3. Пункты с созданием БД и добавлением пользователя

для доступа к ней на второй и последующих репликах можно пропустить, т.к. при первичной инициализации replica set база данных и параметры авторизации будут реплицированы с первичного сервера.

Для обеспечения защищенного внутреннего взаимодействия между репликами необходимо сгенерировать ключевой файл, который после генерации необходимо скопировать на сервер каждой из реплик, независимо от того является ли он Primary, Secondary или Arbiter.

В текущей инструкции считается, что группа серверов состоит из двух реплик БД и арбитра.

Примечание: MongoDB Replica-set может работать и без арбитра, но для быстрой смены Primary в случае падения текущего арбитр необходим. В его отсутствие смена Primary-сервера может занимать от нескольких десятков секунд до нескольких минут. Кроме того, в конфигурациях без арбитра существует ненулевая вероятность потерять часть данных при падении первичного сервера.

2.2.3.1. Подключаемся консольным клиентом:

mongo

Вывод консоли при успешном подключении:

MongoDB shell version v 4.2.17
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.2.17

2.2.3.2. Переключаемся на служебную базу данных:

use admin

Вывод консоли при успешной операции:

switched to db admin

2.2.3.3 Авторизуемся рутом:

db.auth("root","pass")

Вывод консоли при успешной авторизации:

1

2.2.3.4 Добавляем роли, разрешающие управление репликацией для администратора MongoDB:

```
db.grantRolesToUser(
  "root",
   [
     { role: "clusterAdmin", db:"admin"},
     { role: "clusterManager", db:"admin"},
     { role: "clusterMonitor", db:"admin"},
     { role: "hostManager", db:"admin"}
   ]
)
```

2.2.3.5 Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.2.3.6 Генерим ключ при помощи OpenSSL:

```
openssl rand -base64 756 > /opt/mongokey/keyfile
```

2.2.3.7 Меняем разрешения доступа к файлу и владельца. Это ограничение MongoDB для *nix-систем, он не принимает ключевые файлы, доступ к которым возможен для неограниченного числа лиц:

```
chmod 400 /opt/mongokey/keyfile
chown mongod:mongod /opt/mongokey/keyfile
```

2.2.3.8 Открываем файл конфигурации для редактирования:

vi /etc/mongod.conf

2.2.3.9 Добавляем в блок security путь до ключевого файла, раскомментируем секцию #replication и добавляем в нее произвольное имя, которое нужно присвоить replica-set:

```
security:
  authorization: enabled
  keyFile: /opt/mongokey/keyfile
replication:
  replSetName: kndrs
```

2.2.3.10 Перезапускаем сервер MongoDB для принятия изменений конфигурации:

```
systemctl restart mongod
```

2.2.3.11 Копируем сертификат, сгенерированный в п.2.3.6 на вторую и последующие реплики MongoDB, назначаем права доступа к нему, изменяем конфигурацию всех реплик согласно п.п.2.3.7-2.3.10.

2.2.3.12 Проводим инициализацию главного мастер-сервера, с которого будем инициализировать весь кластер. Подключаемся консольным клиентом:

```
mongo
```

Вывод консоли при успешном подключении:

```
MongoDB shell version v 4.2.17
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.2.17
```

2.2.3.13 Переключаемся на служебную базу данных:

```
use admin
```

Вывод консоли при успешной операции:

switched to db admin

2.2.3.14 Авторизуемся рутом:

db.auth("root","pass")

Вывод консоли при успешной авторизации:

1

2.2.3.15 Проверяем текущий статус репликации:

rs.status()

Вывод консоли, со статусом 0 и сообщением NotYetInitialized, означающим, что репликация еще не была инициализирована:

```
{
	"info" : "run rs.initiate(...) if not yet done for the set",
	"ok" : 0,
	"errmsg" : "no replset config has been received",
	"code" : 94,
	"codeName" : "NotYetInitialized",
	"$clusterTime" : {
		"clusterTime" : Timestamp(0, 0),
		"signature" : {
			"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
			"keyId" : NumberLong(0)
		}
	}
}
```

2.2.3.16 Инициализируем главную реплику (замените _id реплик-сета, _id сервера реплики и хост/порт на требуемые в вашей конфигурации):

```
rs.initiate(
  {
    _id : "kndrs",
    members: [
      { _id : 0, host : " 10.206.130.133:27017" }
    ]
  }
)
```

Вывод консоли при успешной инициализации (статус 1):

```
{
        "ok" : 1,
        "operationTime" : Timestamp(1526041434, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1526041434, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

2.2.3.17 Добавляем вторую реплику сета (замените хост/порт на требуемые в вашей конфигурации):

```
rs.add("10.206.130.134:27017")
```

Вывод консоли при успешном добавлении второго сервера в Replica Set:

```
{
	"ok" : 1,
	"operationTime" : Timestamp(1526041890, 1),
	"$clusterTime" : {
		"clusterTime" : Timestamp(1526041890, 1),
		"signature" : {
			"hash" : BinData(0,"OML4OUVIR6f7ROabQcVUS3nvNSs="),
			"keyId" : NumberLong("6554298059960877057")
		}
	}
}
```


2.2.3.18 Добавляем арбитра (замените хост/порт на требуемые в вашей конфигурации):

```
rs.addArb("10.206.130.132:27017")
```

2.2.3.19 Проверяем конфигурацию:

```
rs.status()
```

Вывод консоли - большой JSON с текущей конфигурацией. Нас интересует наличие статусных полей (state), которые могут принимать значение 1, если все ОК и сервер первичный, либо 2, если все ОК и сервер вторичный, либо 7, если все ОК и сервер – арбитр, либо 5, если еще идет синхронизация данных между репликами (расшифровка статуса будет в поле stateStr):


```
{
	"set" : "kndrs",
	"myState" : 1,
	...
	"members" : [
		{
			"_id" : 0,
			"name" : " 10.206.130.133:27017",
```

```
            "health" : 1,
            "state" : 1,
            "stateStr" : "PRIMARY",
                                ...
        },
        {
            "_id" : 2,
            "name" : " 10.206.130.132:27017",
            "health" : 1,
            "state" : 2,
            "stateStr" : "STARTUP2",
            ...
        },

        ...
    ]
    ...
}
```


2.2.3.20 Отключаемся от консоли базы данных:

exit

Вывод консоли при успешной операции:

bye

2.2.4. Просмотр статуса репликации, переконфигурирование replica-set, удаление из него ноды.

Все изменения параметров текущей конфигурации желательно делать на ноде, которая в текущий момент времени является первичной. Но в любом случае, при изменении конфигурации, «первичная» нода теряет свой PRIMARY-статус, и реплика-сет проводит «перевыборы» лидера. Для изменения нод необходимо

авторизоваться в консольном клиенте mongo пользователем, имеющим роль clusterAdmin.

Описанные далее команды изменения настроек нужно выполнять в консольном клиенте mongo на Primary-сервере. Команды просмотра текущего статуса (например, для поиска текущей PRIMARY) можно выполнить на любом сервере. Определить является ли текущая нода PRIMARY можно сразу после запуска консольного клиента, еще до авторизации – в строке приветствия будет выведено имя реплика-сета и статус текущей ноды:

mongo

Вывод консоли:
MongoDB shell version v 4.2.17
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.2.17
kndrs:SECONDARY>

2.2.4.1 Просмотр текущей конфигурации и статуса репликации:

Просмотр статуса репликации:
rs.status()
Просмотр текущей конфигурации:
rs.conf()

2.2.4.2 Изменение параметров текущей конфигурации.

Для изменения параметров текущей конфигурации, без удаления и создания новой ноды (например, изменение хоста расположения ноды, либо его ip-адреса) необходимо выполнить три действия в консольном клиенте mongo
Получаем текущую конфигурацию в качестве переменной:
cfg = rs.conf()
После выполнения команды получения в консоль будет выведена JSON с текущей конфигурацией.
Меняем в ней какие-либо параметры требуемой реплики:

cfg.members[0].host = "newhost.mymongo.ru:27027"

cfg.members[0].priority = 2

Реконфигурируем кластер:

rs.reconfig(cfg)

2.2.4.3 Удаление ноды из кластера

Для удаления ноды достаточно выполнить следующую команду:

rs.remove("newhost.mymongo.ru:27027")

2.3 Установка утилиты полнотекстового поиска ElasticSearch

2.3.1 Установка последней стабильной версии на RHEL 8.

2.3.1.1 Устанавливаем JDK 8:

yum install java-1.8.0-openjdk

Импортируем GPG-ключ:

rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch

2.3.1.2 Создаем файл yum-репозитория со следующим содержимым:

[elasticsearch]

name=Elasticsearch repository for 7.x packages

baseurl=https://artifacts.elastic.co/packages/7.x/yum

gpgcheck=1

gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch

enabled=0

autorefresh=1

type=rpm-md

2.3.1.3 Устанавливаем Elasticsearch командой:

sudo yum install --enablerepo=elasticsearch elasticsearch

2.3.1.4 Добавляем сервис в автозапуск:

systemctl enable elasticsearch

2.3.1.5 В конфиг-файле /etc/elasticsearch/elasticsearch.yml меняем три строки:

- network.host: 0.0.0.0 – слушать на всех интерфейсах, позволяет подключаться к сервису по сети,
- discovery.seed_hosts: ["knd-dbs2t.passport.local"] – прописывается имя нод(ы),
- cluster.initial_master_nodes: ["knd-dbs2t.passport.local"] - прописывается имя нод(ы).

2.3.1.6 Запускаем сервис:

systemctl start elasticsearch

2.3.1.7 Из консоли проверяем запуск следующей командой:

curl http://localhost:9200

Пример вывода при успешном запуске:

{
"name" : "knd-dbs2t.passport.local",
"cluster_name" : "elasticsearch",
"cluster_uuid" : "gTwxHLOfQ4elFkcq3AYuGQ",
"version" : {
"number" : "7.15.2",
"build_flavor" : "default",
"build_type" : "rpm",
"build_hash" : "93d5a7f6192e8a1a12e154a2b81bf6fa7309da0c",
"build_date" : "2021-11-04T14:04:42.515624022Z",
"build_snapshot" : false,
"lucene_version" : "8.9.0",
"minimum_wire_compatibility_version" : "6.8.0",

"minimum_index_compatibility_version" : "6.0.0-beta1"
},
"tagline" : "You Know, for Search"
}

Первые три строки должны быть заполнены.

2.4 Установка сервиса хранения данных - ftp-сервера vsftpd

2.4.1. Установка последней стабильной версии на RHEL 8.

2.4.1.1 Устанавливаем сервис ftp из стандартного репозитория:

yum install vsftpd

2.4.1.2 Добавляем сервис в автозапуск:

systemctl enable vsftpd

2.4.1.3 Заводим пользователя с домашней директорией (ключ -b) и задаем пароль:

adduser ftpuser -b /u01/ftp/

passwd ftpuser

2.4.1.4 В директории с конфигурацией vsftpd создаем файл allowed_users_list, где прописываем имя пользователя (ftpuser), которому будет разрешен доступ по протоколу ftp.

2.4.1.5 В конфигурационном файле vsftpd.conf производим следующие изменения:

- anonymous_enable=NO – запрещает доступ анонимным пользователям,
- local_enable=YES – разрешает доступ локальным пользователям,
- write_enable=YES – разрешает пользователям загружать данные,
- chroot_local_user=YES – запрещает пользователям доступ вне их домашней директории,

- allow_writeable_chroot=YES – разрешает пользователям запись в их домашние директории,
- userlist_enable=YES – позволяет создать список ftp-пользователей,
- userlist_file=/etc/vsftpd/allowed_users_list – список пользователей, которым разрешен доступ на ftp-сервер,
- userlist_deny=NO – разрешает доступ только пользователям из списка allowed_users_list,
- pasv_min_port=2000, pasv_max_port=3000 – диапазон портов, используемых серверов в пассивном режиме.

2.4.1.6 Запускаем сервис vsftpd:

systemctl start vsftpd

2.5. Установка брокера сообщений RabbitMQ

2.5.1 Установка последней стабильной версии на RHEL 8 в режиме standalone

2.5.1.1 Создаем репозиторий yum для RabbitMQ путем исполнения скрипта:

curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.rpm.sh | sudo bash

2.5.1.2. Создаем кэш пакетов из только что добавленного репозитория:

yum -q makecache -y --disablerepo='*' --enablerepo='rabbitmq_erlang' --enablerepo='rabbitmq_server'

2.5.1.3. Устанавливаем необходимые пакеты:

yum install socat logrotate

2.5.1.4. Устанавливаем библиотеку Erlang и брокер сообщений RabbitMQ:

yum install --repo rabbitmq_erlang --repo rabbitmq_server erlang rabbitmq-server –y

2.5.1.5. Добавляем сервис в автозапуск:

systemctl status rabbitmq-server.service

2.5.1.6. Запускаем сервис:

systemctl start rabbitmq-server

2.5.1.7. Проверяем статус:

rabbitmqctl status

Примерный вывод команды (статус должен быть Runtime):

Status of node rabbit@knd-app4p ...

Runtime

OS PID: 3109660

OS: Linux

Uptime (seconds): 163913

Is under maintenance?: false

RabbitMQ version: 3.9.11

Node name: rabbit@knd-app4p

Erlang configuration: Erlang/OTP 24 [erts-12.2] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [jit]

Erlang processes: 1273 used, 1048576 limit

Scheduler run queue: 1

Cluster heartbeat timeout (net_ticktime): 60

2.5.1.8. Включаем плагин управления и мониторинга:

rabbitmq-plugins enable rabbitmq_management

2.5.1.9. Создаем пользователя, которого мы хотим сделать администратором:

rabbitmqctl add_user admin password

2.5.1.10. Сообщаем RabbitMQ, что пользователь admin – администратор:

rabbitmqctl set_user_tags admin administrator

2.5.1.11. Наделяем пользователя admin полными правами:

rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"

2.5.1.12. Удаляем гостевого пользователя guest:

rabbitmqctl delete_user guest

2.5.2. Запуск RabbitMQ в режиме кластера

2.5.2.1 Устанавливаем брокер сообщений на второй сервер. Для этого необходимо выполнить пункты 2.5.1.1 – 2.5.1.8.

2.5.2.2 Копируем содержимое файла /var/lib/rabbitmq/.erlang.cookie с уже установленной ноды на дополнительную:

echo "DLKNCLNDNCSDWEUECICE" | sudo tee /var/lib/rabbitmq/.erlang.cookie

2.5.2.3 Стартуем сервис на второй ноде:

systemctl start rabbitmq-server

2.5.2.4 Проверяем состояние кластера на обеих нодах. В блоке Running Nodes должны быть представлены две ноды:

Running Nodes

rabbit@knd-app3p

rabbit@knd-app4p

2.5.2.5 На второй ноде проверяем репликацию созданного на первой ноде пользователя admin:

rabbitmqctl list_users

Примерный вывод команды:

Listing users ...

user tags

admin [administrator]

Пользователи успешно реплицировались.

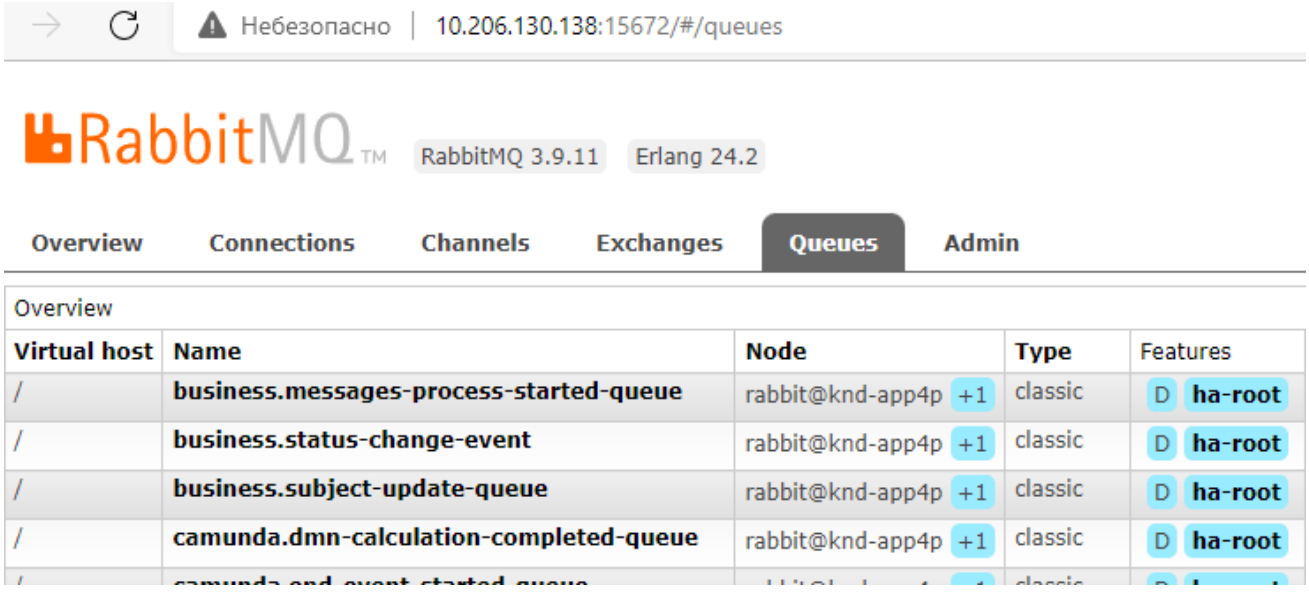
2.5.2.6 Для репликации очередей с первичного на вторичный сервер нужно настроить зеркалирование с помощью политики:

rabbitmqctl set_policy ha-root "" '{"ha-mode":"all","ha-sync-mode":"automatic"}'

После этого в веб-интерфейсе RabbitMQ по адресу:

http://ip-address:15672

в меню Queues в столбце Node будем видно имя первичной ноды для этой очереди и +1. Это означает, что данная очередь зеркалируется еще на один сервер:



2.5.3 Разграничение прав в RabbitMQ

Все шаги выполняются из командной строки Linux из-под пользователя с правами root.

2.5.3.1. Создаем пользователя gisok:

rabbitmqctl add_user gisok password

2.5.3.2. Создаем виртуальный хост:

rabbitmqctl add_vhost gisok_vhost

2.5.3.3. Даем пользователю admin полные права на этот виртуальный хост:

rabbitmqctl set_permissions -p gisok_vhost admin ".*" ".*" ".*"

2.5.3.4. С правами пользователя admin в виртуальном хосте gisok_vhost создаем точку обмена exchange:

rabbitmqadmin declare exchange --vhost=gisok_vhost name=gisok-exchange type=direct -u admin –p adminpassword

2.5.3.5. Для жалоб и КНМС также в виртуальном хосте gisok_vhost создаем 2 входящих и 2 исходящих очереди:

rabbitmqadmin declare queue --vhost=gisok_vhost name=gisok_complaints_in durable=true -u admin –p adminpassword

rabbitmqadmin declare queue --vhost=gisok_vhost name=gisok_complaints_out durable=true -u admin –p adminpassword

rabbitmqadmin declare queue --vhost=gisok_vhost name=gisok_knms_in durable=true -u admin –p adminpassword

rabbitmqadmin declare queue --vhost=gisok_vhost name=gisok_knms_out durable=true -u admin –p adminpassword

2.5.3.6. Чтобы пользователь gisok не смог создать большее количество очередей, для виртуального хоста gisok_vhost устанавливаем их максимальное количество:

```
rabbitmqctl set_vhost_limits -p gisok_vhost '{"max-queues": 4}'
```

2.5.3.7. С помощью ключей маршрутизации (routing_key) привязываем очереди к точке обмена:

```
rabbitmqadmin --vhost="gisok_vhost" declare binding source="gisok-exchange" destination_type="queue" destination="gisok_complaints_in" routing_key="gisok_complaints_in" -u admin –p adminpassword

rabbitmqadmin --vhost="gisok_vhost" declare binding source="gisok-exchange" destination_type="queue" destination="gisok_knms_in" routing_key="gisok_knms_in" -u admin –p adminpassword

rabbitmqadmin --vhost="gisok_vhost" declare binding source="gisok-exchange" destination_type="queue" destination="gisok_complaints_out" routing_key="gisok_complaints_out" -u admin –p adminpassword

rabbitmqadmin --vhost="gisok_vhost" declare binding source="gisok-exchange" destination_type="queue" destination="gisok_knms_out" routing_key="gisok_knms_out" -u admin –p adminpassword
```

2.5.3.8. В виртуальном хосте gisok_vhost даем права пользователю gisok на использование очередей и точки обмена:

```
rabbitmqctl set_permissions -p gisok_vhost gisok "^gisok.*" "^(gisok.*in|gisok-exchange)$" "^(gisok.*out|gisok-exchange)$"
```

2.5.3.9. Предоставляем пользователю gisok право доступа к веб-интерфейсу:

rabbitmqctl set_user_tags gisok management

2.6. Установка СУБД PostgreSQL

2.6.1. Установка версии 9.6 на RHEL 8

2.6.1.1. Устанавливаем PostgreSQL-9.6 командой:

yum install @postgresql:9.6

2.6.1.2. Инициализируем БД:

postgresql-setup --initdb

2.6.1.3. Добавляем сервис в автозапуск:

systemctl enable postgresql.service

2.6.1.4. Запускаем сервис:

systemctl start postgresql.service

2.6.1.5. Устанавливаем пароль для пользователя postgres:

- Переключаемся на пользователя postgres и заходим в консоль psql:
  su - postgres
  psql

- В консоли psql вводим:
  \password postgres

  и устанавливаем пароль.

- Выходим из консоли psql:
  \q

2.6.1.6. Создаем нового пользователя, задаем пароль и БД, назначаем пользователю права на новую БД:

```
su - postgres
createuser --interactive
createdb -O newuser newdb
```

2.6.1.7. В конфигурационном файле /var/lib/pgsql/data/postgresql.conf включаем возможность подключаться к БД по сети:
раскомментируем строку listen_addresses и сделаем ее равной
listen_addresses = '*'

2.6.1.8. В конфигурационном файле /var/lib/pgsql/data/pg_hba.conf меняем local на:

```
local   all             all                                     md5
```

и добавляем:

```
host    all             all             all                     md5
```

2.6.1.9. Перезапускаем сервис:

```
systemctl restart postgresql.service
```

2.7. Развертывание системы в подах Kubernetes

Для упрощения управления и оркестровки набор микросервисов разворачивается в системе, основанной на системе контейнеризации Kubernetes, под названием Rancher. Микросервисы разворачиваются в Rancher соответствии с документацией, а именно:

- Создается Deployment и Service. В процессе создания указывается образ микросервиса; порты, по которым сервис будет доступен в проекте и на которых запущены процессы внутри контейнеров; проверка состояния контейнеров – Health Check и т.д.
- Создается точка входа в приложение – Ingress-сервис, благодаря которому приложение будет доступно извне.

– Настраивается сетевое взаимодействие микросервисов с внешними сервисами: СУБД Mongo, PostgreSQL, FTP, RabbitMQ, ElasticSearch, а также с органами государственной власти.

Все образы строятся из «базового» образа, включающего в себя только JDK8, располагающегося в нем по пути /opt/jdk8 (это сделано для того, чтобы не занимать место на диске копируя в каждый создаваемый образ JDK8, размером в пятьсот с лишним мегабайт).

Базовый образ в предоставляемой конфигурации основан на контейнере Ubuntu 16.04. Ввиду того, что для работы большинства сервисов системы нужен только JDK8, окружение ОС не принципиально - можно использовать любой другой базовый контейнер (Centos, Debian, Alpine и пр.). Для смены базового контейнера на требуемый нужно отредактировать файл /base/Dockerfile, изменив в нем строчку «FROM ubuntu:16.04» изменив имя базового образа на необходимый. Не рекомендуется использовать «пустой» базовый контейнер, т.к. в нем нет утилит для управления содержимым контейнера, которые могут потребоваться для его администрирования.

Файлы конфигурации необходимо передавать в образ в момент его сборки либо в момент старта контейнера в Rancher с помощью объектов ConfigMap и Secret. Преимущество ConfigMap и Secret в том, что есть возможность редактирования конфигурационных файлов без пересборки образа и повторного разворачивания контейнера с сервисом конфигурации.