Ассистент для команд разработки

Инструкция по развертыванию и установке

Всего листов: 54

Аннотация

Настоящий документ представляет собой руководство для развертывания и установки Ассистента для команд разработки (АКР). Руководство содержит подробное описание подготовки инфраструктуры (включая LLM, объектное хранилище и Kubernetes), развертывания компонентов системы (интерфейс АКР, микросервисы DWH), а также инструкции по сборке исходных кодов и проверке работоспособности компонентов. Документ предназначен для администраторов, осуществляющих внедрение и настройку АКР.

Содержание

1. Введение

1.1.Область применения системы

Областью применения Системы является деятельность по написанию программного кода, тестированию ПО и написанию технической и сопровождающей документации для разработки ПО.

1.2.Уровень подготовки пользователя

Для выполнения операций, предусмотренных настоящим Руководством, пользователю с ролью «Администратор» потребуются знания основ информационной безопасности, навыки администрирования серверов и вычислительных сетей, рекомендуется наличие навыков и опыта работы с технологиями docker, kubernetes, Keycloak, Postgresql. Необходимо обладать опытом управления системами с искусственным интеллектом (ИИ) и опытом настройки LLM, что предполагает наличие навыков создания, редактирования и оптимизации инструкций для LLM, навыков оценки качества выходных данных LLM, навыков анализа текстов для составления инструкций и примеров для LLM. Также потребуется понимание принципов обработки естественного языка, понимание синтаксиса, семантики и языковой структуры русского и английского языков.

1.3.Перечень эксплуатационной документации для ознакомления

Перед началом работы по установке, настройке, администрированию Системы пользователю с ролью «Администратор» необходимо ознакомиться с настоящим Руководством в полном объеме.

1.4.Глоссарий

В тексте настоящего документа представлены следующие сокращения (см. **Таблица 1-1**).

Таблица 1-1 *— Перечень сокращений*

Сокращение/аббревиатура	Значение
АРМ	Автоматизированное рабочее место
ИИ	Искусственный интеллект
ПО	Программное обеспечение
Система	Ассистент для команд разработки
СУБД	Система управления базой данных

Сокращение/аббревиатура	Значение
ТЗ	Техническое задание
LLM	Large Language Model – Большая языковая модель

В тексте настоящего документа представлены следующие термины и определения (см. **Таблица 1-2**).

Таблица 1-2 *— Перечень терминов и определений*

Термин	Определение
Авторизация	Предоставление определенному лицу или группе лиц прав на выполнение определенных действий
Аутентификация	Проверка подлинности предъявленного Пользователем идентификатора
Администратор Системы	Пользователь Системы с ролью «Администратор»
Заказчик	Физическое или юридическое лицо, которое инициирует, заказывает и финансирует разработку, модификацию или поддержку программного обеспечения Системы. После приемки услуг по разработке Системы Заказчик является Владельцем Системы
Идентификатор Пользователя	Уникальный признак объекта, позволяющий отличать его от других
Идентификация	Процедура, в результате выполнения которой для субъекта выявляется его идентификатор, однозначно определяющий этого субъекта в Системе интеллектуального анализа документов
Компонент (программного обеспечения)	Составная часть программного обеспечения, выполняющая определенную функцию
Пользователь	Работник, наделенный правами доступа к информационным ресурсам организации
Подсказка	Системный промпт для LLM - представляет собой руководящие инструкции или исходные данные, которые задаются модели, чтобы она могла генерировать ответы на запросы пользователя
Ролевая модель	Распределение прав доступа и обязанностей между ролями. Определяет, какие действия может выполнять каждая роль в рамках работы с Системой, включая создание и редактирование правил, проведение проверок и управление

Роль	Набор полномочий, который необходим Пользователю для выполнения определённых рабочих задач. Каждый сотрудник может иметь одну или несколько ролей, а каждая роль может содержать от одного до множества полномочий, которые разрешены Пользователю в рамках этой роли. Роли Пользователей в Системе: – «Пользователь»; – «Администратор».
ИТ-инфраструктура	Совокупность всего программного обеспечения, оборудования, сетей и подключенных сервисов, образующих ИТ-среду организации
Файл	Цифровой носитель информации, содержащий текстовые и(или) графические данные, с любым из перечисленных расширений: .txt, .java, .js, .ts, .html, .MD.

2. Общее описание системы «Ассистент для команд разработки»

2.1.Назначение системы

Система предназначена для ускорения процесса написания программного кода за счет ответов на вопросы, улучшения качества кода через анализ ошибок и рекомендаций по улучшению, сокращения времени разработки и вывода продуктов на рынок, сокращения материальных затрат на разработку новых сервисов.

В целях помощи командам разработки Система предназначена для удовлетворения задач использования безопасных решений, соответствующих стандартам работы в корпоративных контурах, включая поддержку замкнутых сетей и исключение утечек данных.

Система подразумевает возможность подбора размера LLM под существующую ИТ-инфраструктуру и потребности Заказчика.

Система подразумевает её использование в качестве веб-приложения или по API другими приложениями, такими как плагины для интегрированных сред разработки (IDE).

В Системе предусматривается использование инструментов для оценки эффективности решения и мониторинга активности пользователей.

Система предназначена для организаций, занимающихся разработкой и поддержкой программных продуктов с большим количеством команд разработки, и направлена на повышение безопасности и эффективности процессов разработки, анализа и тестирования ПО..

2.2.Условия применения системы

2.2.1.Общие условия

Режим эксплуатации Системы: 5x8 (5 дней в неделю по 8 часов) в часы работы Заказчика.

Для возможности администрирования Системы должны выполняться условия, предусмотренные пп. 2.2.2, 2.2.3, 2.2.4 Руководства.

Для настройки журналирования событий по информационной безопасности уровень логирования рекомендуется выставлять в warning - как компромиссный уровень между объемом информации и детализацией.

2.2.2.Требования к инфраструктуре

Для работы Системы требуется 5 виртуальных машин, состав которых определен в **Таблице 2-1** настоящего Руководства.

Таблица 2-1 – *Состав виртуальных машин*

Компонент	Состав	Название сервера	ip	Примечание

Центр анализа – LLM	code LLM, nvidia GPU drivers, docker-toolkit docker engine	vdc-rk-llm-gpu	10.10.10.10	-
СУБД	PostgreSQL	vdc-rk-postgres01	10.10.10.11	-
Master node kubernetes 01	k8s	vdc-rk-k8s01	10.10.10.12	-
Worker node kubernetes 02	k8s	vdc-rk-k8s02	10.10.10.13	-
Сервер объектного хранилище S3	minio	vdc-rk-minio01	10.10.10.19	-

2.2.3.Требования к рабочей станции для развертывания контура

Для выполнения операций по развертыванию контура Системы на рабочей станции должны быть выполнены следующие требования:

- операционная система Linux:
  - рабочая станция должна быть запущена на базе операционной системы (ОС) Linux;
  - ОС должна взаимодействовать с драйверами на GPU;
  - дополнительные требования к ОС определяются внутренними документами Заказчика;
- программное обеспечение:
  - helm;
  - kubectl;
  - sshpass;
- сетевые настройки: рабочая станция должна иметь доступ по SSH ко всем серверам, участвующим в развертывании контура Системы;
- доступ по SSH-ключам: для упрощения процесса развертывания рекомендуется настроить беспарольный доступ по SSH с рабочей станции на все серверы;
- права доступа: пользователь на рабочей станции должен иметь привилегированный доступ (sudo) для выполнения административных задач;

- настройки безопасности: SSH-ключи, используемые для доступа к серверам, должны быть защищены и безопасно храниться;
- дополнительные утилиты: утилиты `wget`, `curl`, `tar`, `unzip`, `telnet nc net-tools` должны быть подготовлены для скачивания и распаковки файлов;
- резервное копирование: рекомендуется регулярно создавать резервные копии конфигурационных файлов и ключей на рабочей станции;
- проверка и тестирование: на рабочей станции должны быть установлены инструменты для тестирования и проверки конфигураций, такие как ping, telnet, nslookup, net-tools, nc;
- документация: рабочая станция должна иметь доступ к актуальной документации о Системе и инструкциям по развертыванию.

2.2.4.Требования к программному обеспечению инфраструктуры

Для обеспечения работы Системы необходимо соблюдать следующие требования:

- наличие программного обеспечения (ПО) на виртуальных машинах (VM), составляющих инфраструктуру Системы, соответствующего требованиям, установленным в п. 2.2.4 Руководства;
- ОС: поддерживает драйверы GPU Nvidia H100 и соответствует техническим требованиям Заказчика по RHEL;
- требования к виртуальным машинам – предоставляются отдельно;
- в контуре должен быть установлен NTP-сервер;
- все машины контура должны синхронизировать время;
- на всех хостах должен быть заведён служебный пользователь с правом выполнения команд с привилегиями суперпользователя для установки и настройки компонентов Системы;
- на всех хостах должен быть установлен и настроен ssh с возможностью авторизации служебным пользователем;
- CPU – рекомендуется Xeon Gold или аналог;
- отсутствие VPN соединений между VM;
- сеть:
    - скорость не менее 1 Гбит;
    - агрегация физических линков на серверах для отказоустойчивости (не

менее 2 подключений);

- для схемы размещения в одном ЦОД все серверы должны быть в одной подсети без фильтрации трафика между ними;

- на каждой виртуальной машине должен быть установлен и настроен iptables;
- наличие Nexus: в Nexus должны быть созданы по списку репозитории для apt, helm, npm, maven, docker, pypi;
- наличие docker для запуска контейнеризованных приложений, необходимый для работы с docker-образами и контейнерами.

3. Инструкция по развертыванию системы «Ассистент для команд разработки»

3.1.Подготовка инфраструктуры к развертыванию системы

3.1.1.Порядок разворачивания LLM (vdc-rk-gpu)

3.1.1.1.Общие сведения

Для разворачивания необходимо последовательно выполнить следующие шаги:

a) Создать машину в VMware vCloudDirector с названием vdc-rk-gpu на основе ОС Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-94-generic x86_64) и с характеристиками, приведенными в **Таблице 2-1** настоящего Руководства.

b) При загрузке ОС убедиться, что версия ядра – ОС 5.15.0-94-generic.

c) Выполнить настройку сетевых интерфейсов, указав IP-адрес 10.10.10.10.

d) Создать отдельного не привилегированного пользователя «adduser user-worker» для выполнения настроек.

e) Добавить пользователя в группу sudo с использованием команды:

```
sudo usermod -aG sudo user-worker
```

f)

3.1.1.2.Установка драйвера для nVidia GPU

Для установки необходимо последовательно выполнить следующие шаги:

a) Проверить подключение видеокарт к серверу перед установкой драйвера с использованием команды:

```
lspci -nnk
```

b)

Ожидаемый результат – вывод списка видеокарт, подключенных к серверу.

a) При наличии видеокарт выполнить установку драйверов.

3.1.1.3.Установка NVIDIA CUDA drivers

Установку следует осуществлять на основании официальной документации, размещенной в сети Интернет по ссылке: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/.

3.1.1.4.Проверка GPU

Для проверки наличия графического процессора с поддержкой CUDA GPU необходимо в

командной строке указать команду:

```
lspci | grep -i nvidia
```

Если не отображаются какие-либо параметры, то следует обновить базу данных PCI-оборудования, которая ведется в Linux, для чего необходимо:

a) Указать «update-pciids» в командной строке (обычно находится в /sbin).

b) Повторно выполнить предыдущую команду «lspci».

Примечание – GPU поддерживает CUDA в случае, если видеокарта – NVIDIA и указана в списке, размещенном в сети Интернет по ссылке: https://developer.nvidia.com/cuda-gpus).

3.1.1.5.Проверка версии Linux

Для проверки наличия поддерживаемой версии Linux (определение установленного дистрибутива и номера релиза) необходимо указать команду в командной строке:

```
uname -m &amp;&amp; cat /etc/*release
```

Примечание – средства разработки CUDA поддерживаются только в некоторых определенных дистрибутивах Linux; перечислены в примечаниях к выпуску CUDA Toolkit.

Ожидаемый результат должен быть похож на указанный в примере ниже, и изменен для конкретной системы.

Пример

```
x86_64
Red Hat Enterprise Linux Workstation выпуск 6.0 (Santiago)
```

Примечание – строка x86_64 указывает на то, что работа выполняется на 64-битной системе. Остальная часть содержит информацию о дистрибутиве.

3.1.1.6.Установка CUDA и проверки установки

Установку следует осуществлять на основании официальной документации, размещенной в сети Интернет по ссылке: https://developer.nvidia.com/cuda-downloads?target_os =Linux&target_arch=x86_64&Distribution=Ubuntu & target_version = 22.04&target_type=deb_ local.

3.1.1.7.CUDA Toolkit Installer

Для установки необходимо последовательно выполнить следующие шаги:

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu
2204/x86_64/cuda-ubuntu2204.pin

sudo mv cuda-ubuntu2204.pin /etc/apt/preferences.d/cuda-
repository-pin-600

wget
https://developer.download.nvidia.com/compute/cuda/12.6.2/local
_installers/cuda-repo-ubuntu2204-12-6-local_12.6.2-560.35.03-
1_amd64.deb

sudo dpkg -i cuda-repo-ubuntu2204-12-6-local_12.6.2-560.35.03-
1_amd64.deb

sudo cp /var/cuda-repo-ubuntu2204-12-6-local/cuda-*-keyring.gpg
/usr/share/keyrings/sudo apt-get update

sudo apt-get -y install cuda-toolkit-12-6
```

3.1.1.8.Установка драйвера

Для установки необходимо последовательно выполнить следующие шаги:

a) По установке с использованием команды:

```
sudo apt-get install -y nvidia-open
```

b)

b) Перезагрузить Систему, чтобы загрузить драйверы NVIDIA:

```
sudo reboot
```

c)

c) Проверить корректность установки драйверов – в терминале выполнить команду nvidia-smi.

Ожидаемый результат – отображение в таблице описания всех видеокарт.

3.1.1.9.Установка docker engine

Для установки необходимо последовательно выполнить следующие шаги:

a) Добавить репозиторий docker:

```
# Add Docker's official GPG key:
 sudo apt-get update
 sudo apt-get install ca-certificates curl
 sudo install -m 0755 -d /etc/apt/keyrings
 sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -
o /etc/apt/keyrings/docker.asc
 sudo chmod a+r /etc/apt/keyrings/docker.asc


 # Add the repository to Apt sources:
 echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release &amp;&amp; echo "$VERSION_CODENAME")
stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
 sudo apt-get update
```

b)

b) Обновить список пакетов в репозиториях:

```
sudo apt-get update
```

c)

c) Установить необходимые пакеты:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

d)

d) Добавить пользователя в группу docker:

```
sudo usermod -aG docker $USER
```

e)

3.1.1.10.Установка и настройка Container toolkit

Для настройки и установки необходимо последовательно выполнить следующие шаги:

a) Настроить репозиторий:

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey
| sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-
toolkit-keyring.gpg \
&& curl -s -L https://nvidia.github.io/libnvidia-
container/stable/deb/nvidia-container-toolkit.list | \
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-
container-toolkit-keyring.gpg] https://#g' | \
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

b)

b) Обновить список пакетов в репозиториях:

```
sudo apt-get update
```

c)

c) Установить NVIDIA Container Toolkit пакеты:

```
sudo apt-get install -y nvidia-container-toolkit
```

d)

Примечание – действия следует выполнять через Apt.

3.1.1.11.Тестирование установки драйверов и компонентов для работы с GPU в docker

Для проверки того, что GPU передает в docker-контейнеры необходимо запустить тестовый контейнер с использованием команды:

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Ожидаемый результат:

```
+-----------------------------------------------------------------------------+
 | NVIDIA-SMI 535.86.10 Driver Version: 535.86.10 CUDA Version:
12.2 |
 |-------------------+----------+----------+
 | GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC
|
 | Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute
M. |
 | | | MIG M. |

|===================+======================+===============|
 | 0 Tesla T4 On | 00000000:00:1E.0 Off | 0 |
 | N/A 34C P8 9W / 70W | 0MiB / 15109MiB | 0% Default |
 | | | N/A |
 +-------------------+----------+-------------+

 +-----------------------------------------------+
 | Processes: |
 | GPU GI CI PID Type Process name GPU Memory |
 | ID ID Usage |

|=============================================================|
 | No running processes found |
 +-------------------------------------------------------------+
```

Отображение ожидаемого результата означает, что установка завершена, драйверы работают корректно.

После окончания установки можно выполнить действия по развертыванию центра анализа.

Примечание – действия следует осуществлять на основании официальной документации, размещенной в сети Интернет по ссылке: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/sample-workload.html.

3.1.2.Порядок разворачивания СУБД (vdc-rk-postgres01)

3.1.2.1.Общие сведения

Для разворачивания необходимо последовательно выполнить следующие шаги:

a) Создать машину в VMware vCloudDirector с названием vdc-rk-postgres01 на основе ОС Ubuntu 22.04.3 LTS и с характеристиками, приведенными в **Таблице 2-1** настоящего Руководства.

b) Выполнить настройку сетевых интерфейсов, указав IP-адрес 10.10.10.11.

c) Создать отдельного не привилегированного пользователя «adduser user-worker» для выполнения настроек.

d) Добавить пользователя в группу sudo.

3.1.2.2.Установка PostgreSQL

Для установки необходимо последовательно выполнить следующие шаги:

a) Установить Postgresql 14 из предоставленных дистрибутивов.

b) Проверить установку и запуск PostgreSQ с использованием команды:

```
sudo systemctl start postgresql
sudo systemctl enable postgresql
sudo systemctl status postgresql
```

c)

3.1.2.3.Создание базы данных, пользователя, схемы и предоставления привилегий

Пример создания базы данных, пользователя, схемы и предоставления привилегий приведен ниже.

Пример

Необходимо последовательно выполнить следующие шаги:

a) Подключиться к PostgreSQL от имени пользователя postgres:

```
sudo -i -u postgres psql
```

b)

b) Создать пользователя и базу данных akr_project.

c) Подключиться к созданной базе данных:

```
CREATE USER akr_project WITH PASSWORD 'password';
CREATE DATABASE akr_project;
\c akr_project;
```

d)

d) Создать схему.

e) Предоставить все привилегии пользователю akr_project:

```
CREATE SCHEMA akr_project;
GRANT ALL PRIVILEGES ON DATABASE akr_project TO akr_project;
GRANT ALL PRIVILEGES ON SCHEMA akr_project TO akr_project;
```

```
ALTER SCHEMA akr_project OWNER S TO akr_project;
```

f)

Примечание – создание остальных баз данных по аналогии с graph_project.

3.1.3.Порядок разворачивания объектного хранилища S3 (vdc-rk-minio)

Для разворачивания необходимо последовательно выполнить следующие шаги:

a) Создать машину в VMware vCloudDirector с названием vdc-rk-minio на основе ОС Ubuntu 22.04.3 LTS и с характеристиками, приведенными в **Таблице 2-1** настоящего Руководства.

b) Выполнить настройку сетевых интерфейсов, указав IP адрес 10.10.10.19.

c) Создать отдельного не привилегированного пользователя adduser user-worker для выполнения настроек.

d) Добавить пользователя в группу sudo.

3.1.3.1.Добавление прав sudo пользователю

Для добавления прав необходимо в файле /etc/sudoers.d/user-worker добавить и сохранить следующую строку:

```
user-worker ALL=(ALL) NOPASSWD: ALL
```

3.1.4.Установка одиночного экземпляра MinIO

3.1.4.1.Загрузка и установка MinIO

Для загрузки и установки необходимо последовательно выполнить следующие шаги:

a) Скачать исполняемый файл MinIO:

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

b)

b) Сделать файл исполняемым:

```
chmod +x minio
```

c)

c) Переместить файл в каталог /usr/local/bin/ для удобного запуска:

```
sudo mv minio /usr/local/bin/
```

d)

3.1.4.2.Создание директории для данных и конфигурации

Для создания директории необходимо последовательно выполнить следующие шаги:

a) Создать директорию, где будут храниться данные MinIO:

```
sudo mkdir /media/minio-data
```

b)

b) Создать пользователя и группу для MinIO:

```
sudo useradd -r minio-user -s /sbin/nologin
```

c)

c) Назначить созданному пользователю права на директорию:

```
sudo chown -R minio-user:minio-user /mnt/data
```

d)

3.1.4.3.Настройка MinIO как службы

Для настройки необходимо последовательно выполнить следующие шаги:

a) Создать конфигурационный файл службы:

```
sudo nano /etc/systemd/system/minio.service
```

b)

b) Вставить в файл следующий контент:

```
[Unit]
 Description=MinIO Object Storage
 After=network.target

 [Service]
 User=minio-user
 Group=minio-user
 ExecStart=/usr/local/bin/minio server /media/minio-data --
console-address ":9001"
 Restart=always
 LimitNOFILE=65536
 Environment="MINIO_ROOT_USER=admin"
 Environment="MINIO_ROOT_PASSWORD=strongpassword"

 [Install]
 WantedBy=multi-user.target
```

c)

c) Сохранить файл.

d) Закрыть файл.

e) Включить и осуществить запуск службы:

```
sudo systemctl daemon-reload
sudo systemctl enable minio
sudo systemctl start minio
```

f)

3.1.4.4.Проверка статуса MinIO

Для проверки статуса запуска необходимо использовать команду:

```
sudo systemctl status minio
```

3.1.4.5.Доступ к веб-интерфейсу MinIO

После запуска можно открыть веб-браузер и перейти по ссылке: http://10.10.10.19:9001.

Необходимо использовать учетные данные, указанные в файле службы:

- имя пользователя: admin;
- пароль: strongpassword.

3.1.5.Порядок разворачивания kubernetes

3.1.5.1.Общие сведения

Для разворачивания необходимо последовательно выполнить следующие шаги:

a) Создать машину в VMware vCloudDirector с названием vdc-rk-k8s01 на основе ОС Ubuntu 22.04.3 LTS и с характеристиками, приведенными в **Таблице 2-1** настоящего Руководства.

b) Выполнить настройку сетевых интерфейсов, указав IP-адрес 10.10.10.12.

c) Создать отдельного не привилегированного пользователя adduser user-worker для выполнения настроек.

d) Добавить пользователя в группу sudo.

3.1.5.2.Подготовка VM к установке kubernetes

3.1.5.2.1.Обновление операционной системы

Необходимо обновить все пакеты Системы до актуальных версий:

```
sudo apt update && sudo apt upgrade -y
```

3.1.5.2.2.Настройка имени хоста и hosts-файла

Для функционирования кластера в нормальном режиме следует настроить уникальное имя хоста для каждого узла.

Также необходимо:

a) Добавить запись в файл /etc/hosts для разрешения имен узлов:

```
sudo nano /etc/hosts
```

b)

b) Добавить следующую строку (изменить IP и имя хоста на необходимые значения):

```
10.10.10.12 vdc-rk-k8s01
10.10.10.13 vdc-rk-k8s02
```

c)

3.1.5.2.3.Отключение swap

Для временного отключения swap необходимо использовать команду:

```
sudo swapoff -a
```

Для постоянного отключения swap закомментируйте или удалите строку, содержащую swap, в файле /etc/fstab:

```
sudo nano /etc/fstab
```

Примечание – необходимо отключить swap, т.к. kubernetes не поддерживает работу с включенным swap.

3.1.5.3.Настройка сетевых параметров – включение iptables для перенаправления трафика

Для включения необходимо последовательно выполнить следующие шаги:

a) Проверить, что параметры netfilter позволяют пересылать трафик:

```
sudo modprobe br_netfilter
```

b)

b) Создать конфигурационный файл:

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf br_netfilter
EOF
```

c)

c) Включить необходимые параметры:

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf net.bridge.bridge-
nf-call-ip6tables = 1 net.bridge.bridge-nf-call-iptables = 1
EOF
```

d)

d) Принять изменения:

```
sudo sysctl --system
```

e)

Примечание – для kubernetes необходима корректная работа iptables.

3.1.5.4.Установка docker

Для установки необходимо последовательно выполнить следующие шаги:

a) Добавить репозиторий docker:

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc


# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release &amp;&amp; echo "$VERSION_CODENAME")
stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get update
```

b)

b) Обновить список пакетов в репозиториях:

```
sudo apt-get update
```

c)

c) Установить необходимые пакеты:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

d)

d) Добавить пользователя в группу docker:

```
sudo usermod -aG docker $USER
```

e)

Примечание – для kubernetes необходимо наличие контейнерной среды для запуска контейнеров. Можно использовать docker, а также и другие, например, containerd.

3.1.5.5.Настройка конфигурации docker для работы с cgroup

Для корректной работы с kubernetes следует настроить cgroup driver. Для этого необходимо последовательно выполнить следующие шаги:

a) Создать или изменить файл /etc/docker/daemon.json:

```
sudo nano /etc/docker/daemon.json
```

b)

b) Добавить следующие параметры:

```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
  "max-size": "100m"
  },
  "storage-driver": "overlay2"
 }
```

c)

c) Применить изменения и перезапустить docker:

```
sudo systemctl restart docker
```

d)

3.1.5.6.Открытие необходимых портов

Для корректного функционирования кластера kubernetes необходимо открыть следующие порты:

- на master -узле:
  - 6443: API сервер kubernetes;
  - 2379-2380: etcd серверы;
  - 10250: kubelet API;
  - 10251: kube-scheduler;
  - 10252: kube-controller-manager;
- worker -узле:
  - 10250: kubelet API
  - 30000-32767: NodePort для доступа к сервисам.

Также следует открыть порты в firewall:

```
sudo ufw allow 6443
sudo ufw allow 2379:2380/tcp
sudo ufw allow 10250
sudo ufw allow 10251
sudo ufw allow 10252
sudo ufw allow 30000:32767/tcp
```

3.1.5.7.Синхронизация времени

Для корректной работы кластера необходимо, чтобы время на всех узлах было

синхронизировано.

Для установки NTP необходимо выполнить:

```
sudo apt install -y ntp
sudo systemctl enable ntp sudo systemctl start ntp
```

3.1.5.8.Установка необходимых инструментов

3.1.5.8.1.Проверка утилит

Для проверки установки на сервере необходимых утилит следует выполнить:

```
sudo apt install -y apt-transport-https ca-certificates curl
```

3.1.5.8.2.Настройка доступа по ssh между всеми машинами по ключу

Для настройки необходимо последовательно выполнить следующие шаги:

a) Выполнить генерацию SSH-ключей на управляющей машине – команду для создания SSH-ключей на машине, которая будет использоваться для развертывания:

```
ssh-keygen -t ed25519 -C "коментарий по поводу ключа"
```

b)

Примечание – по умолчанию ключи будут сохранены в файле ~/.ssh/id_ed25519. При создании можно оставить поля пароля пустыми, чтобы не вводить его каждый раз при подключении.

b) Выполнить копирование публичного ключа на все узлы в кластере с использованием команд ssh-copy-id:

```
ssh-copy-id user-worker@10.10.10.12
ssh-copy-id user-worker@10.10.10.13
```

c)

Ожидаемый результат – отображено предложение ввести пароль для каждого узла, после чего публичный ключ будет добавлен в файл ~/.ssh/authorized_keys на каждом узле, а также можно подключаться к узлам по ключам без пароля.

3.1.5.8.3.Проверка подключения по SSH

Для проверки возможности подключения к каждому узлу без ввода пароля после копирования ключей необходимо выполнить:

```
ssh user-worker@10.10.10.13
```

В случае успешного подключения можно выйти с использованием команды exit и

повторить действие для остальных узлов.

3.1.5.8.4.Проверка обратного подключения между узлами

Для проверки возможности подключения узлов друг к другу можно повторить процесс генерации ключей и копирования на каждом узле или в ручном режиме добавить публичные ключи каждого узла в файл ~/.ssh/authorized_keys на других узлах.

3.1.5.8.5.Добавление прав sudo пользователю

Для добавления прав необходимо в файле /etc/sudoers.d/user-worker добавить и сохранить следующую строку:

```
user-worker ALL=(ALL) NOPASSWD: ALL
```

3.1.5.8.6.Иные условия

Необходимо повторить все вышеуказанные настройки на каждой VM под k8s кластер.

3.1.5.9.Установка kubernetes

3.1.5.9.1.Подготовка виртуальных машин и создание *SnapShot*

Перед началом развертывания кластера необходимо проверить соответствие VM следующим требованиям:

- на всех машинах установлена ОС Ubuntu 22.04;
- серверы настроены и выполнено отключение swap;
- выполнена настройка сеть;
- обеспечен доступ по SSH с одного узла на другие.

Внимание! Перед началом установки следует сделать *SnapShot* всех виртуальных машин через vCloud Director. Для этого необходимо последовательно выполнить следующие шаги:

a) Зайти в интерфейс vCloud Director.
b) Перейти к требуемой виртуальной машине.
c) Выбрать Create Snapshot.
d) Проверить, что созданный SnapShot можно использовать для восстановления в случае возникновения проблем.
e) Повторить процесс для всех машин, которые будут участвовать в кластере.

3.1.5.9.2.Подготовка машины для развертывания Kubespray

Для установки kubernetes с использованием kubespray необходимо наличие одной машины (локальная или отдельная VM), с которой будет выполняться развертывание.

3.1.5.9.3.Установка необходимых утилит на машине для развертывания

Для установки необходимо выполнить:

```
sudo apt update &amp;amp;&amp;amp; sudo apt install -y python3-pip git
sshpass
```

3.1.5.9.4.Клонирование репозитория kubespray

Для клонирования репозитория kubespray необходимо выполнить:

```
git clone https://github.com/kubernetes-sigs/kubespray.git
 cd kubespray
```

3.1.5.9.5.Установка зависимостей

Для установки требуемых Python-зависимостей необходимо выполнить:

```
sudo pip3 install -r requirements.txt
```

3.1.5.9.6.Настройка инвентаря

Для настройки необходимо скопировать пример инвентарного файла, а также выполнить его настройку:

```
cp -rfp inventory/sample inventory/rks-cluster
```

3.1.5.9.7.Генерация файла hosts.yaml

Необходимо использовать скрипт inventory builder для автоматического создания конфигурационного файла:

```
declare -a IPS=(10.10.10.12 10.10.10.13)
 CONFIG_FILE=inventory/rks-cluster/hosts.yaml python3
contrib/inventory_builder/inventory.py ${IPS[@]}
```

Также следует в ручном режиме выполнить настройку роли узлов (например, master node и worker node), отредактировать файл inventory/rks-cluster/hosts.yaml:

```
all:
  hosts:
  vdc-rk-k8s01:
  ansible_host: 10.10.10.12
  vdc-rk-k8s02:
  ansible_host: 10.10.10.13
  children:
  kube_control_plane:
  hosts:
  vdc-rk-k8s01:
  kube_node:
  hosts:
  vdc-rk-k8s02:
  etcd:
  hosts:
  vdc-rk-k8s01:
  k8s_cluster:
  children:
  kube_control_plane:
  kube_node:
  calico_rr:
  hosts: {}
```

3.1.5.9.8.Настройка конфигурации kubespray

Для настройки необходимых параметров следует отредактировать файл конфигурации inventory/rks-cluster/group_vars/all/all.yml:

```
kube_network_plugin: calico
```

3.1.5.9.9.Отключение установки docker в kubespray

В случае, если docker уже установлен, то следует отключить установку docker и зависимостей через kubespray. Для этого необходимо последовательно выполнить следующие шаги:

a) Открыть файл inventory/rks-cluster/group_vars/k8s_cluster/docker.yml.

b) Найти или добавить параметр для передачи kubespray команды не выполнять установку docker, если он уже установлен на узле:

```
docker_skip_install: true
```

c)

3.1.5.9.10.Запуск процесса развертывания

Для запуска необходимо последовательно выполнить следующие шаги:

a) Проверить доступность всех серверов для ansible:

```
ansible all -i inventory/rks-cluster/hosts.yaml -m ping
```

b)

b) Выполнить команду:

```
ansible-playbook -i inventory/rks-cluster/hosts.yaml --become --become-user=user-worker cluster.yml
```

c)

Примечание – опция become позволяет Ansible выполнять команды с привилегиями root.

3.1.5.9.11.Проверка состояния кластера

Для проверки корректности работы кластера после завершения установки необходимо последовательно выполнить следующие шаги:

a) Подключиться к master node:

```
ssh user-worker@vdc-rk-k8s01
```

b)

b) Выполнить команду для проверки состояния узлов:

```
kubectl get nodes
```

c)

Ожидаемый результат – вывод списка из четырех узлов, что подтверждает факт разворачивания kubernetes.

c) Можно проверить, что все компоненты Системы установлены и работают, с использованием команды:

```
kubectl get pod -A
```

d)

Ожидаемый результат – вывод список всех подов Системы во всех namespace и статус у каждого пода – Running.

3.1.5.10.Установка local-path StorageClass

Для использования local-path необходимо проверить local-path Storagев в кластере. Для этого необходимо выполнить следующие шаги:

a) Создайть манифест local-path-storage.yaml:

```
apiVersion: storage.k8s.io/v1
 kind: StorageClass
 metadata:
  name: local-path
 provisioner: rancher.io/local-path
 reclaimPolicy: Retain
 volumeBindingMode: WaitForFirstConsumer
```

b)

b) Применить его:

```
kubectl apply -f local-path-storage.yaml
```

c)

3.2.Развертывание компонентов системы

3.2.1.Развертывание компонента интерфейса АКР

3.2.1.1.Компоненты, которые должны быть развернуты в kubernetes

Компоненты, которые должны быть развернуты в kubernetes, приведены в **Таблице 3-1** настоящего Руководства.

Таблица 3-1 *– Компоненты, которые должны быть развернуты в kubernetes*

Name	Namespace
backend-api	rks-akr
backend-auth	rks-akr
backend-project	rks-akr
backend-report-service	rks-akr
backend-s3-adapter	rks-akr
backend-search	rks-akr
akr-ui	rks-akr

Каждый микросервис загружается в kubernetes с помощью docker-образа микросервиса и yaml файла с описанием объектов данного сервиса для kubernetes.

Готовые docker-образы находятся в папке /_ _, которые необходимо загрузить в docker registry. Для этого в папке следует запустить скрипт upload_images_to_registry.sh.

Пример скрипта приведен ниже.

Пример

```
#!/bin/bash
docker load < backend-api.tar
docker load < backend-auth.tar
docker load < backend-project.tar
docker load < backend-report-service.tar
docker load < backend-s3-adapter.tar
docker load < backend-search.tar
docker load < akr-ui.tar

docker push localhost:5000/backend-api:latest
docker push localhost:5000/backend-auth:latest
docker push localhost:5000/backend-project:latest
docker push localhost:5000/backend-report-service:latest
docker push localhost:5000/backend-s3-adapter:latest
docker push localhost:5000/backend-search:latest
docker push localhost:5000/akr-ui:latest
```

3.2.1.2.Развертывание сервисов в kubernetes

Для развертывания необходимо последовательно выполнить следующие шаги:

a) Создать namespace:

```
kubectl create namespace rks-akr
```

b)

b) Для развертывания всех сервисов запустить команду:

```
kubectl apply -f mirrion/
```

c)

Примечание – в папке akr/ находятся файлы манифестов для развертывания объектов kubernetes.

c) Через некоторое проверить работу подов всех микросервисов в namespace rks-akr:

```
kubectl get pod -n rks-akr
```

d)

Ожидаемый результат – статус Running.

3.2.1.3.Дополнительная инструкция по развертыванию компонентов АКР

Список необходимых сервисов приведен ниже:

- *Backend-сервисы:*
  - *Backend-api;*
  - *Backend-project;*
  - *Backend-search;*
  - *Backend-s3-adapter;*
- *Frontend-сервис: akr-ui.*

Пример файлов конфигурации и рекомендуемые значения параметров для манифестов в k8s приведены ниже.

Пример

- *специфические параметры при настройке ingress отсутствуют;*
- *при инициализации проекта необходимо запускать сервисы в следующей последовательности: маркировок, метаданных, API, далее последовательность не имеет значение;*
- *примерное потребление ресурсов приведено в* ***Таблице 3-2*** *настоящего Руководства.*

Таблица 3-2 *– Примерное потребление ресурсов*

Сервис	Мин. ОЗУ	Мин.ЦПУ
Backend-api	1Gb	0.3m
Backend-project	2Gb	0.3m
Backend-search	1Gb	0.3m
AKR-websocket	0.5Gb	0.3m
Backend-s3-adapter	1Gb	0.3m
Backend-auth	0.5Gb	0.3m
akr-ui	256Mb	0.1m

При установке сервисов в Linux как демонов (daemon), необходимо. выгрузить jar-файлы и создать юниты, в которых будет описан их запуск/перезапуск.

Пример юнита приведен ниже.

Пример

```
[Unit]
Description=Java App AKR
[Service]
Environment="JASYPT_ENCRYPTOR_PASSWORD=graph-backend"
EnvironmentFile=/path/to/envfile
 #путь к env файлу для каждого сервиса свой
ExecStart=/usr/bin/java -jar /path/to/your/app.jar
 #путь к java файлу приложения
User=in-asu-akr
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

При использовании манифестов в k8s для каждого сервиса должны быть созданы секреты, конфигпамы, сервисы и деплоймент конфигурации. Создание рассмотрено ниже на примере сервиса backend-api:

Примеры

1. ***Манифест configmap.yaml:***

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-api-config
  namespace: rks-dev
data:
  STORE_PASSWORD: "changeit"
  DATABASE_URL: "jdbc:postgresql://10.10.10.10:5432/akrui"
  DATABASE_USERNAME: "akrui"
  PROJECT_NAME: "AKR"
  ASYNC_DATA_LIFETIME: "5"
  GROUP_ID: "backend-api-dev-group"
  ASYNC_RESULT_TOPIC: "result-topic-dev"
  FIND_PATH_TOPIC: "find-path-dev"
  APPLICATION_NAME: "backend-api-dev"
  SWAGGER_URL: "http://10.10.10.10:8081/api"
 COLLECTION_HISTORY_DAYS: "100"
  COLLECTION_HISTORY_CRON: "0 0 */2 * * *"
```

2.

2. ***Манифест deployment.yaml:***

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-api
  labels:
    app: backend-api
  namespace: akr-dev
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: backend-api
  template:
    metadata:
      labels:
        app: backend-api
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - backend-api
            topologyKey: "kubernetes.io/hostname"
      tolerations:
      - key: "node.kubernetes.io/disk-pressure"
        operator: "Exists"
        effect: "NoSchedule"
      containers:
      - name: backend-api
        image: 10.10.10.10:30005/akr-docker-
snapshot/graph/ui/backend-api:latest
        ports:
        - containerPort: 8080
        # volumeMounts:
        # - name: keystore-volume
```

```
    #   mountPath: "/etc/keystore"
    #   readOnly: true
    # - name: ssl-volume
    #   mountPath: "/etc/pki/ca-trust/extracted/java"
    #   readOnly: true
    resources:
      limits:
        cpu: "2.5"
        memory: "2048Mi"
      requests:
        cpu: "1.4"
        memory: "1024Mi"
    livenessProbe:
      httpGet:
        path: /actuator/health
        port: 8080
      initialDelaySeconds: 60
      periodSeconds: 20
      timeoutSeconds: 10
      failureThreshold: 5
    readinessProbe:
      httpGet:
        path: /actuator/health
        port: 8080
      initialDelaySeconds: 30
      periodSeconds: 20
      timeoutSeconds: 10
      successThreshold: 1
    envFrom:
    - configMapRef:
        name: backend-api-config
    - configMapRef:
        name: common-setting-config
    - secretRef:
        name: backend-api-secret
  # volumes:
  # - name: keystore-volume
  #   secret:
  #     secretName: keystore-secret
  # - name: ssl-volume
```

```
    #   secret:
    #     secretName: ssl-secret
```

3.

3. Манифест secret.yaml:

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-api-secret
  namespace: akr-dev
type: Opaque
stringData:
  DATABASE_PASSWORD: "strong_password_for_example_12332145"
  ARANGO_PASSWORD: "strong_password_for_example_12332145"
  JASYPT_ENCRYPTOR_PASSWORD: "graph-backend"
```

4.

4. Манифест srvc.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: backend-api-service
  namespace: akr-dev
  labels:
    app: backend-api
spec:
  selector:
    app: backend-api
  ports:
  - protocol: TCP
    port: 9090
    nodePort: 31001
    targetPort: 8080
  type: NodePort
```

5.

5. Манифест с общими настройками для всех сервисов:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: common-setting-config
  namespace: akr-dev
data:
  SSL_ENABLE: "FALSE"
  STORE_PATH: "/etc/keystore/keystore.p12"
  STORE_PASSWORD: "changeit"
  TRUST_STORE_PATH: "/etc/keystore/keystore.p12"
  TRUST_STORE_PASSWORD: "changeit"
  KEY_ALIAS: "graph"

  PORT: "8080"

  DATABASE_TIMEOUT: "30000"
  DATABASE_MAX_CONNECTIONS: "15"

  KIBANA_ENABLED: "FALSE"

  SPRING_ACTIVE_PROFILES: "keycloak"
  PROFILE: "keycloak"

  WEBSOCKET_TOPIC: "websocket-dev"
  PREGEL_REGISTER_JOB_TOPIC: "pregel-register-job-dev"
  GRAPH_INFLUENCE_REGISTER_JOB: "graph-influence-register-job-
dev"
  PREGEL_SERVICE_NAME: "pregel-service-dev"
  RESET_TASK_CALLBACK_TOPIC: "reset-task-callback-dev"

  URL_BACKEND_API: "http://backend-api-service:9090"
  URL_AUTH_SERVICE: "http://backend-auth-service:9090"
  URL_PROJECT_SERVICE: "http://backend-project-service:9090"
  URL_PREGEL_SERVICE: "http://backend-pregel-service:9090"
  URL_BACKEND_SERVICE: "http://backend-api-service:9090"
  URL_ASYNC_SERVICE: "http://backend-async-gateway-
service:9090"

  GRAPH-CORE-API: "http://backend-api-service:9090"
  KAFKA_CONCURRENCY: "4"
```

6.

Все остальные манифест-файлы для других сервисов находятся в папке поставки РК СВИД – Исходные коды / Пользовательский интерфейс / k8s-manifest.

Сведения о переменных окружения приведены в **Таблице 3-3** настоящего Руководства.

Таблица 3-3 *– Переменные окружения*

Название параметра	Пример значения	Описание
SSL_ENABLE	TRUE/FALSE	Включение SSL шифрования сервера (https протокол)
STORE_PATH	/opt/back/ssl/keystore.p12	Путь к хранилищу, содержащему приватный ключ сервера для SSL протокола и публичные доверенные ключи (сертификаты). Хранилище в формате p12
STORE_PASSWORD	123	Пароль от хранилища приватных/публичных ключей заданных в STORE_PATH
KEY_ALIAS	graph	Алиас ключа, импортированного в хранилище приватных/публичных ключей заданных в STORE_PATH. Ссылка: https://security.stackexchange.com/questions/123944/what-is-the-purpose-role-of-the-alias-attribute-in-java-keystore-files
PORT	8080	Порт, на котором будет запущено приложение (сервер)
DATABASE_TIMEOUT	30000	Таймаут соединения к реляционной базе данных (в мс)
DATABASE_MAX_CONNECTIONS	30	Максимально возможное количество одновременных соединений к реляционной базе данных

Название параметра	Пример значения	Описание
DATABASE_URL	jdbc: postgresql://localhost:6432/graphui	Адрес соединения с реляционной базой данных в формате jdbc (ссылка: https://docs.oracle.com/cd/E17952_01/connector-j-8.0-en/connector-j-reference-jdbc-url-format.html)
DATABASE_USERNAME	user	Пользователь реляционной базы данных, от имени которого будет выполнятся соединение
DATABASE_PASSWORD	123 или зашифрованный ENC (rtuouq+neOBt+BPq1JdyD01cBxmlxhjOfFEj27Hqbys=)	Пароль от пользователя (DATABASE_USERNAME) реляционной базы данных. Может быть в открытом или зашифрованном виде. Шифруется через jasypt (ссылка – https://www.devglan.com/online-tools/jasypt-online-encryption-decryption)
PROJECT_NAME	ORG	Наименование заказчика проекта. У каждого может быть свой. Влияет на наполнение базы данных. Под каждого заказчика – отдельное наполнение, зависящее от этого параметра
PROFILE	sfera	Режим запуска приложения
TRUST_STORE_PATH	/opt/back/ssl/keystore.p12	Путь к хранилищу публичных ключей (сертификатов) для защищенного соединения к arangodb

Название параметра	Пример значения	Описание
TRUST_STORE_PASSWORD	123	Пароль от хранилища публичных ключей (сертификатов) для защищенного соединения к arangodb
JASYPT_ENCRYPTOR_PASSWORD	my-secret	Кодовое слово для шифрования паролей методом jasypt. secret key (по ссылке – https://www.devglan.com/online-tools/jasypt-online-encryption-decryption)
URL_AUTH_SERVICE	https://10.1.1.1:8082	http адрес микросервиса аутентификации. backend-auth
ALLOWED_ORIGINS	*	-
KIBANA_ENABLED	false	Включение/выключение логов в json формате
APPLICATION_NAME	backend-search	Наименование микросервиса
USER_FIO_FIELD	givenName	Наименование поля из JWT токена, из которого сервис отчетов будет извлекать ФИО (можно не заполнять, по умолчанию given_name)
GRAPH-CORE-API	http://backend-api-service:9090	Http url сервиса backend-ap

3.2.5.Развертывание сервиса DWH

Для сервиса dwh требуется наличие PostgreSQL (развертывание выполнено на предыдущих шагах).

В архиве с исходными кодами необходимо запустить скрипт для развертывания базы и заполнения справочников под сервис dwh (/db/ivd_dwh_init.sql).

Примечание – в файле указана процедура запуска файла по частям от одного пользователя, далее – часть от другого пользователя.

Также потребуется создание базы данных внутри Arangodb.

Пример скрипта для создания базы приведен ниже.

Пример

```
sudo arangosh   # enter root (arangodb) password



#Create database.
db._createDatabase("graph_db");
db._createDatabase("project_db");
db._createDatabase("graph_metadata");
db._createDatabase("report_db");
db._createDatabase("graphui");



#Create user
var users = require("@arangodb/users");
users.save("dwh-client", "2KMaKFtA2V6yfZax");



#Grant access to the database created above.
users.grantDatabase("dwh-client", "graph_db");
users.grantDatabase("dwh-client", "project_db");
users.grantDatabase("dwh-client", "graph_metadata");
users.grantDatabase("dwh-client", "report_db");
users.grantDatabase("dwh-client", "graphui");



# list databases
db._databases()



# check connection
arangosh --server.username "dwh-client" --server.database
"graph_db"
```

Для создания базы в PostgreSQL необходимо запустить SQL-скрипт:

a) Создать пользователя и базу данных ivd_dwh:

```
CREATE USER ivd_etl_user WITH PASSWORD '';
CREATE DATABASE ivd_dwh;
\c ivd_dwh;
CREATE SCHEMA ivd_dwh;
GRANT ALL PRIVILEGES ON DATABASE ivd_dwh TO givd_etl_user;
GRANT ALL PRIVILEGES ON SCHEMA ivd_dwh TO ivd_etl_user;
```

```
ALTER SCHEMA ivd_dwh OWNER TO ivd_etl_user;
```

b)

b) Создать пользователя и базу данных graph_mark:

```
CREATE USER graph_mark_user WITH PASSWORD '';
CREATE DATABASE graph_mark;
\c graph_mark;
CREATE SCHEMA graph_mark;
GRANT ALL PRIVILEGES ON DATABASE graph_mark TO graph_mark_user;
GRANT ALL PRIVILEGES ON SCHEMA graph_mark TO graph_mark_user;
ALTER SCHEMA graph_mark OWNER TO graph_mark_user;
```

c)

c) Создать пользователя и базу данных graph_project:

```
CREATE USER graph_project_user WITH PASSWORD '';
CREATE DATABASE graph_project;
\c graph_project;
CREATE SCHEMA graph_project;
GRANT ALL PRIVILEGES ON DATABASE graph_project TO
graph_project_user;
GRANT ALL PRIVILEGES ON SCHEMA graph_project TO
graph_project_user;
ALTER SCHEMA graph_project OWNER TO graph_project_user;
```

d)

d) Создать пользователя и базу данных graphui:

```
CREATE USER graphui_user WITH PASSWORD '';
CREATE DATABASE graphui;
\c graphui;
CREATE SCHEMA graphui;
GRANT ALL PRIVILEGES ON DATABASE graphui TO graphui_user;
GRANT ALL PRIVILEGES ON SCHEMA graphui TO graphui_user;
ALTER SCHEMA graphui OWNER TO graphui_user;
```

e)

e) Создать пользователя и базу данных graph_metadata:

```
CREATE USER graph_metadata_user WITH PASSWORD '';
CREATE DATABASE graph_metadata;
\c graph_metadata;
CREATE SCHEMA graph_metadata;
GRANT ALL PRIVILEGES ON DATABASE graph_metadata TO
graph_metadata_user;
GRANT ALL PRIVILEGES ON SCHEMA graph_metadata TO
graph_metadata_user;
ALTER SCHEMA graph_metadata OWNER TO graph_metadata_user;
```

f)

3.2.7.Развертывание LLM

3.2.7.1.Репозитории с кодом

Необходимо скачать в полученном архиве репозитории с кодом и разархивировать.

3.2.7.2.Сборка и запуск компонентов

Запуск компонентов должен осуществляться путем сборки docker-контейнеров.

3.2.7.3.Задание переменных окружения для получения конфигов из FTP, доступа к LLM и БД

Для того, чтобы задать креды для подключения в S3 при запуске компонентов в docker-контейнере необходимо создать .env файл в корневом каталоге с исходными кодами:

```
# Имя пользователя для доступа к FTP
FTP_USERNAME=<username>
# Пароль пользователя для доступа к FTP
FTP_PASSWORD=<password>
# Имя пользователя для доступа к внешней БД системы ИВД
EXT_DB_USERNAME=<username>
# Пароль пользователя для доступа к внешней БД системы ИВД
EXT_DB_PASSWORD=<username>
# Ключ доступа к API большой языковой модели (LLM)
# см. раздел Установка и запуск vLLM
LLM_API_KEY=<secret_key>
# Путь до конфигурационного файла system_check
PATH_CONFIG_SYSTEMCHECK=<ftp://hostame:port/путь/до/файла-настройки-systemcheck.yaml>
# Путь до конфигурационного файла akr
PATH_CONFIG_AKR=<ftp://hostame:port/путь/до/файла-настройки-akr.yaml>
# Путь до каталога хранения лог файлов компонента.
# Не указывайте / в конце пути
# Компонент сохранит в нем файл с именем akr.log и system_check.log
# Не указывайте имя файла в пути
PATH_LOG_DIR=</путь/до/каталога/логов>
# Путь до модели - укажите абсолютный путь
PATH_TO_MODEL=</путь/до/модели>
```

3.2.7.4.3Запуск компонентов из docker-контейнеров

Необходимо собрать и запустить контейнеры, используя docker-compose файл:

```
docker compose up -d --build
```

3.2.7.5.3Запуск и установка vLLM

Необходимо выполнить:

a) Разместить файлы модели из дистрибутива поставки Системы на машине, предназначенной для хостинга vLLM.

Примечание – путь до модели указывается в параметре model при старте vLLM.

b) Установить vLLM на машине, предназначенной для хостинга vLLM.

Внимание! Необходимо установить следующие зависимости в отдельном окружении python:

```
pip install transformers==4.43.3 vllm==0.5.3.post1
```

c) Установить требуемые драйверы графических процессоров на машину, предназначенную для хостинга vLLM.

Примечание – в случае использования графических процессоров NVIDIA H100 необходимо устанавливать следующие драйверы: `Driver Version: 545.23.08  CUDA Version: 12.3`.

d) Создать для работы vLLM пользовательский сервис linux – vllm.service:

```
nano ~/.config/systemd/user/vllm.service
```

e)

e) Вставить в файл следующее содержимое:

```
[Unit]
Description=vLLM API Server
After=network.target

[Service]
Environment=PATH=/home/dsuser/.local/bin:/usr/local/bin:/usr/bin:/bin
Type=simple
ExecStart=/usr/bin/python3 -m vllm.entrypoints.openai.api_server \
    --port 3423 \
    --model <path/to/model> \
    --api-key <sectet_api_key> \
    --tensor-parallel-size 2 \
    --disable-custom-all-reduce \
    --uvicorn-log-level debug
Restart=always
RestartSec=3

[Install]
WantedBy=default.target
```

f)

f) Указать путь до каталога с моделью – <path/to/model> в параметре PATH_TO_MODEL env файла (описание приведено в п. 3.2.7.3 настоящего Руководства, а также в конфигурационном файле system_check.yaml (раздел common_params: llm_connection: model)).

g) Указать ключ для доступа к API vLLM – <sectet_api_key> в параметре LLM_API_KEY env файла (описание приведено в п. 3.2.7.3 настоящего Руководства):

```
chmod 644 ~/.config/systemd/user/vllm.service
systemctl --user daemon-reload
```

```
systemctl --user enable vllm.service
```

h)

h) Проверить статус:

```
systemctl --user status vllm.service
```

i)

i) Посмотреть логи:

```
journalctl --user -u vllm.service
```

j)

j) Остановить:

```
systemctl --user stop vllm.service
```

k)

k) Рестарт:

```
systemctl --user restart vllm.service
```

l)

l) Для старта сервиса даже без логина пользователя:

```
loginctl enable-linger $USER
```

m)

4. Инструкция по сборке исходных кодов системы «Ассистент для команд разработки»

4.1. Запуск компонентов непосредственно в среде разработки LLM

Для запуска АКР необходимо выполнить:

```
uvicorn akr_app:app --host 0.0.0.0 --port 9061 --workers 4
```

Для запуска akr_consumer необходимо разместить конфигурационный файл из src/iad_repo/config/akr.yaml в бакете S3, а также выполнить:

```
sh start_consumer.sh akr_consumer.py
s3://<bucket_name>/path/to/akr.yaml logs/akr.log
```

Последним аргументом передается путь до лог-файла компонента.

Скрипт start_consumer.sh дополнительно вызывает скрипт start_ssh.sh в котором создается ssh-proxy с подключением к серверу с Postgres.

Для запуска System_check необходимо выполнить:

```
uvicorn system_check_app:app --host 0.0.0.0 --port 9062 --
workers 4
```

Для System_check_consumer необходимо разместить конфигурационный файл из src/iad_repo/config/system_check.yaml в бакете S3, а также выполнить:

```
sh start_consumer.sh system_check_consumer.py
s3://<bucket_name>/path/to/system_check.yaml
logs/system_check.log
```

Последним аргументом передается путь до лог-файла компонента.

Скрипт start_consumer.sh дополнительно вызывает скрипт start_ssh.sh, в котором создается ssh-proxy с подключением к серверу с Postgres.

4.2. Проверка работы компонентов

4.2.1. Общие сведения

Документация по API компонентов АКР и System_check находится по адресам:

- http://localhost:9061/docs – akr;
- http://localhost:9062/docs – System_check.

Для проверки работы компонента akr_consumer необходимо выполнить подключение к

БД:

```
host: localhost
port: 5432
username: ivd_etl_user
password: nTxBTAmrMoTrQn4a
database: ivd_dwh
```

Перед подключением к БД должен быть запущен ssh прокси (скрипт start_ssh.sh).

4.2.2.Проверка system_check_consumer.py

Необходимо добавить в таблицу event.evnt_rgtr новое событие (запись):

```
INSERT INTO "event".evnt_rgtr (
     evnt_type_id, doc_id, evnt_context, created_at,
     created_by_code, evnt_type_name
 ) values (
    3, null,
    '[{"attribute_id": 1, "profile_id": 0}]',
    LOCALTIMESTAMP, '', 'req_system_check'
 );
```

Далее следует ожидать новую запись в event.evnt_rgtr с evnt_type_name="system_check_event".

5. Журнал автотестирования системы «Ассистент для команд разработки»

Журнал автотестирования Системы заполняется в соответствии с формой, которая представлена в **Таблице 5-1** настоящего Руководства.

Таблица 5-1 *– Форма журнала автотестирования*

Дата тестирования	Наименование теста	Результат теста	Комментарий