ИСП РАН

«TALISMAN»

ТЕХНИЧЕСКОЕ ОПИСАНИЕ

Оглавление

1 Системные требования

Все компоненты (серверная часть аналитической подсистемы и подсистемы сбора, базы данных, подсистемы аудита, поиска и пр.) программы поставляются в виде Docker-контейнеров.

Развертывание программы выполняется с помощью открытого ПО Kubernetes, позволяющего выполнять эту операцию на нескольких узлах с возможностью дальнейшего автоматического масштабирования сервисов для оптимизации скорости работы конвейера обработки данных. Для развертывания и функционирования программы необходимо ПО Docker и Kubernetes (в т.ч. настроенный для работы с GPU, ingress и presistent volumes).

Функционирование программы гарантируется при развертывании с помощью под Docker версии 19.03.12 и Kubernetes версии 1.18.9, работающих под операционной системой Ubuntu (или аналогичных ОС Linux). Работа с реализациями Docker и Kubernetes под другими операционными системами не гарантируется.

1.1 Минимальные требования к аппаратным средствам

- 512 Гб ОЗУ,
- 2 видеокарты, аналогичных Nvidia Tesla T4,
- Intel Xeon Gold 6144, 3,5 ГГц, 8 ядер (или аналогичный) в количестве 4 штук (или аналогичное количество потоков),
- 105 Тб дискового пространства.

2 Развертывание системы

2.1 Подготовка инфраструктуры для установки

2.1.1 Список используемого ПО с указанием лицензий

№	Название ПО	Лицензия	Открытый репозиторий
1	ca-certificates	Mozilla Public License Version 2.0	https://packages.ubuntu.com/focal/ca-certificates
2	containerd	Apache License 2.0	https://github.com/containerd/containerd/blob/main/LICENSE
3	docker-registry	Apache License 2.0	https://docs.docker.com/registry/
4	helm	Apache License 2.0	https://github.com/helm/helm/blob/main/LICENSE
5	ingress	Apache License 2.0	https://github.com/kubernetes/ingress-nginx/blob/main/LICENSE
6	kubectl	Apache License 2.0	https://github.com/kubernetes/kubectl/blob/master/LICENSE
7	Kubernetes	Apache License 2.0	https://github.com/kubernetes/kubernetes/bl

			ob/master/LICENSE
8	kubespray	Apache License 2.0	https://github.com/kubernetes-sigs/kubespray/blob/master/LICENSE
9	nfs-provisioner	Apache License 2.0	https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/blob/master/LICENSE
10	nfs-server	GNU GPL v2	https://packages.ubuntu.com/focal/nfs-common
11	skopeo	Apache License 2.0	https://github.com/containers/skopeo/blob/main/LICENSE

2.1.2 Установка зависимостей

1) сделать sudo для пользователя-установщика без пароля (на всех машинах):

ввести команду sudo visudo, в появившемся файле добавить следующую строку

```
<user> ALL=(ALL) NOPASSWD: ALL
```

2) установить ca-certificates (менеджером пакетов), python >= 3.8 (https://docs.python.org/3/using/unix.html) и pip (https://pip.pypa.io/en/latest/installation/)

3) установить skopeo (https://github.com/containers/skopeo/blob/main/install.md):

- Ubuntu 20.4

```
. /etc/os-release
echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_${VERSION_ID}/ /" | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_${VERSION_ID}/Release.key | sudo apt-key add -
sudo apt-get update
sudo apt-get -y install skopeo
```

- Ubuntu 20.10 и новее

```
sudo apt-get -y update
sudo apt-get -y install skopeo
```

- CentOS 7.x и старше

```
sudo yum -y install skopeo
```

- CentOS 8.x и новее

```
sudo dnf -y install skopeo
```

4) установить helm:

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
bash
```

2.1.3 Создание кластера Kubernetes

Источник - https://github.com/kubernetes-sigs/kubespray

1) установить kubespray:

```
git clone https://github.com/kubernetes-sigs/kubespray.git
cd kubespray
sudo pip3 install -r requirements.txt
cp -rfp inventory/sample inventory/mycluster
```

2) настроить параметры kubespray (соответствующие файлы в inventory/mycluster):

helm (https://github.com/kubernetes-sigs/kubespray/blob/master/inventory/sample/group_vars/k8s_cluster/addons.yml#L7),

ingress + host_network (https://github.com/kubernetes-sigs/kubespray/blob/master/inventory/sample/group_vars/k8s_cluster/addons.yml#L95),

docker-registry:

- helm

```
inventory/mycluster/group_vars/k8s_cluster/addons.yml
# Helm deployment
helm_enabled: true
```

- ingress + host_network

```
inventory/mycluster/group_vars/k8s_cluster/addons.yml
# Nginx ingress controller deployment
ingress_nginx_enabled: true
ingress_nginx_host_network: true
```

3) при развёртывании с количеством узлов, большим одного, настроить docker-registry и обеспечить ssh-доступ с текущего (первого) узла на все остальные (в том числе и на себя) без пароля:

- docker-registry (при развёртывании на кластере не с одним узлом)

```
inventory/mycluster/group_vars/k8s-cluster/addons.yml
# Registry deployment
registry_enabled: true
    создать файл /etc/containers/registries.conf.d/registry.conf со
    следующим содержимым для insecure доступа skopeo
/etc/containers/registries.conf.d/registry.conf
[[registry]]
insecure = true
location = "registry.kube-system.svc.cluster.local:5000"
```

- ssh:

Мастер узел

```
mkdir ~/.ssh
ssh-keygen -t rsa -N "" -f ~/.ssh/master_rsa
# для каждого узла
scp ~/.ssh/master_rsa.pub <node_user>@<node_ip>:~/master_rsa.pub
```

Все узлы

```
mkdir ~/.ssh
echo master_rsa.pub >> ~/.ssh/authorized_keys
```

4) задать IP-адреса узлов и развернуть кластер (может понадобиться sudo или изменение пользователя):

```
declare -a IPS=(<IP адреса узлов>)
CONFIG_FILE=inventory/mycluster/hosts.yaml python3
contrib/inventory_builder/inventory.py ${IPS[@]}
ansible-playbook -i inventory/mycluster/hosts.yaml --become --become-
user=root cluster.yml
```

5) добавить полученный конфиг в .kube

```
mkdir ~/.kube && sudo cp /etc/kubernetes/admin.conf ~/.kube/config && sudo
chown <username>:<usergroup> ~/.kube/config
```

2.1.4 Установка хранилища (в качестве примера: nfs-server и provisioner)

1) установить nfs-server (на все узлы):

- установить nfs-server **(на узел-хранилище)** (при монтировании к внешнему nfs-серверу пропустить):

  Ubuntu

  ```
  sudo apt update
  sudo apt install nfs-kernel-server
  ```

  CentOS 7.x and older

  ```
  yum install nfs-utils
  systemctl enable rpcbind nfs-server
  systemctl start rpcbind nfs-server
  ```

  CentOS 8.x and newer

  ```
  dnf install nfs-utils
  systemctl start nfs-server.service
  systemctl enable nfs-server.service
  systemctl status nfs-server.service
  ```

- установить nfs-client **(на все узлы, кроме узла-хранилища):**

  Ubuntu

  ```
  sudo apt update
  ```

```
sudo apt install nfs-common
```

CentOS 7.x and older

yum install nfs-utils

```
systemctl enable rpcbind
systemctl start rpcbind
```

CentOS 8.x and newer

```
dnf install nfs-utils
```

2) создать директорию для nfs (при монтировании к внешнему nfs-серверу пропустить):

```
sudo mkdir /nfs
sudo chmod -R 777 /nfs
```

3) настроить монтирование nfs, добавив в /etc/exports следующую строчку (* – доступ любому клиенту, no_root_squash – можно добавить, чтобы избежать проблемы с PG (https://github.com/docker-library/postgres/issues/361)) (при монтировании к внешнему nfs-серверу пропустить):

```
/nfs  *(rw,insecure,sync,no_subtree_check,no_root_squash)
```

4) выполнить команду для переинициализации конфигурации nfs (при монтировании к внешнему nfs-серверу пропустить):

```
exportfs -r
```

5) добавить разрешающие правила брандмауэра, если он запущен (при монтировании к внешнему nfs-серверу пропустить):

```
firewall-cmd --permanent --zone=public --add-service=nfs
firewall-cmd --permanent --zone=public --add-service=mountd
firewall-cmd --permanent --zone=public --add-service=rpc-bind
firewall-cmd --reload
```

6) установить nfs-provisioner и сделать это хранилище хранилищем по умолчанию (nfs.server и nfs.path – из /etc/exports или из параметров внешнего хранилища):

```
helm repo add nfs-subdir-external-provisioner https://kubernetes-
sigs.github.io/nfs-subdir-external-provisioner/
helm install --create-namespace -n nfs-client nfs-subdir-external-
provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner
--set nfs.server=x.x.x.x --set nfs.path=/nfs --set
storageClass.archiveOnDelete=true --set storageClass.default=true --set
nodeSelector.kubernetes\\.io/hostname=node1
```

7) Проверить настройку `nfs:` выполнить `kubectl get sc`

рядом с nfs-client будет надпись (default), если нет, то выполнить:

```
kubectl patch storageclass nfs-client -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

2.1.5 Установка средства управления кластером Kubernetes

1) На каждом узле должен быть установлен kubectl (инструкция по установке - https://kubernetes.io/ru/docs/tasks/tools/install-kubectl/#установка-с-помощью-встроенного-пакетного-менеджера) и сформирован /home/<username>/.kube/config, если /home/<username>/.kube/config нет:

```
mkdir -p .kube
cp /etc/kubernetes/admin.conf ~/.kube/config
```

2) если нужно включить автодополнение kubectl:

- Ubuntu

apt-get install bash-completion

```
echo 'source <(kubectl completion bash)' >>~/.bashrc
```

- CentOS 7.x и старше

```
yum install bash-completion
echo 'source <(kubectl completion bash)' >>~/.bashrc
```

- CentOS 8.x и новее

```
dnf install bash-completion
echo 'source <(kubectl completion bash)' >>~/.bashrc
```

2.1.6 Настройка DNS

Настроить DNS так, чтобы нужному url соответствовал нужный ip адрес. Также убедиться, что на вычислительном узле это доменное имя также ресолвится.

2.1.7 Настройка OSM

(необязательно) OSM: если необходимо иметь в системе локальный osm сервер (например, если нет доступа к внешним серверам карт) нужно импортировать данные в osm и nominatim, для этого необходимо добавить в setup-local.yaml определение хранилищ с путями до .pbf и .poly (необязательно). Если необходимо использовать фиксированные теги osm (OSM_TAG) и nominatim (NOM_TAG) нужно указать их в values.global. Если необходимо установить ограничения на память jvm:

osm

```
volumes:
  poly-file:
    path_bundle: <.poly path>
```

```
  pbf-file:
    path_bundle: <.pbf path>
values:
  global:
    OSM_IMPORT: true
    OSM_TAG: <OSM_TAG>
    NOM_TAG: <NOM_TAG>
```

2.2 Развертывание основного приложения

Развертывание приложения Талисман состоит из 2 частей: настройки развертывания для стенда и установки самого приложения из бандла приложения

2.2.1 Подготовка setup-local.yaml

<username> – имя пользователя, от имени которого производится установка; если пользователь root, то /home/<username> → /root

1) положить {} в /home/<username>/.docker/config.json

2) определиться с типом skopeo_push_registry:

- при использование кластера поверх crio или containerd: containers-storage:docker-reference
- при использовании кластера поверх docker: docker-daemon:docker-reference
- при использовании собственного registry в случае более чем одного узла в кластере: docker://docker-reference

3) заполнить setup-local.yaml :

- KUBECONFIG - путь до файла доступа к kubernetes кластера
- KD_DOMAIN - домен на котором разворачивается приложение
- SUBDOMAIN - поддомен, если приложение не на основном домене
- KD_INGRESS_IP - ip адреса ingress контроллера
- DOMAIN_CERT_PATH - https сертификат для домена
- DOMAIN_KEY_PATH - ключ для сертификата домена
- SUBDOMAIN_CERT_PATH - https сертификат для поддомена
- SUBDOMAIN_KEY_PATH - ключ для сертификата поддомена
- DOCKER_AUTH_CONFIG_PATH - путь до файла авторизации в docker-registry
- skopeo_push_registry (при наличии docker-registry в к skopeo_push_registry docker://registry.kube-system.svc.cluster.local:5000)

- imageRegistryInternal, imagePullPolicyExternal (при наличии docker-registry в к указывать registry.kube-system.svc.cluster.local:5000)

```
env-defaults:
  DOCKER_AUTH_CONFIG_PATH: /home/<username>/.docker/config.json
  DOMAIN_CERT_PATH: /home/<username>/bundle/certs/fullchain.cer
  DOMAIN_KEY_PATH: /home/<username>/bundle/certs/<domain>.key
  KD_DOMAIN: <domain>
  KD_INGRESS_IP: <IP adress>
  KUBECONFIG: /home/<username>/.kube/config
  SUBDOMAIN: ''
  SUBDOMAIN_CERT_PATH: /home/<username>/bundle/certs/fullchain.cer
  SUBDOMAIN_KEY_PATH: /home/<username>/bundle/certs/<domain>.key
skopeo_push_registry: "docker://registry:5000/prefix"
values:
  Talisman-pvc:
    copy:
      render: false
    global:
      imagePullPolicyExternal: Always
      imageRegistryExternal: registry:5000/prefix
  global:
    imagePullPolicy: Always
    imagePullPolicyExternal: Always
    imageRegistryExternal: registry:5000/prefix
    imageRegistryInternal:registry:5000/prefix
```

2.2.2 Установка и обновление основных подсистем

Приложение представлено в виде бандла (bundle-<номер версии>.tar.gz):

1) Разместить файл с бандлом в директорию /home/<username>/bundle-<номер версии>.tar.gz

2) Сохраните старую версию бандла командой:

```
mv /home/<username>/bundle /home/<username>/bundle-<номер предыдущей версии>
```

3) Распакуйте архив командой:

```
tar zxvf /home/<username>/bundle-<номер версии>.tar.gz -C /home/<username>/bundle
```

4) Остановите стенд командной

```
bash /home/<username>/bundle/rm.sh -s /home/<username>/setup-local.yaml
```

5) Запустите новую версию стенда командой

```
bash /home/<username>/bundle/up.sh -s /home/<username>/setup-local.yaml
```

2.3 Удаление основных подсистем

Для удаления основных подсистем программы требуется:

1) Остановить стенд командой:

```
bash /home/<username>/bundle/rm.sh -s /home/<username>/setup-local.yaml
```

2) Удалить папку /data и /home/<username>/bundle.

2.4 Возможные проблемы и способы их решения

1) При развёртывании kubespray появляется ошибка Module nf_conntrack_ipv4 not found:

1. ввести команду: modprobe nf_conntrack_ipv4 – если завершается без ошибок, повторить развёртывание
2. ввести команду: modprobe nf_conntrack – если завершается без ошибок выполнить обратные изменения кода https://github.com/kubernetes-sigs/kubespray/pull/6988/files, а именно заменить в файле roles/kubernetes/node/tasks/main.yml Modprobe nf_conntrack_ipv4 и Persist ip_vs modules на:

```
- name: Modprobe nf_conntrack_ipv4 for kernels < 4.19
  modprobe:
    name: nf_conntrack_ipv4
    state: present
  register: enable_nf_conntrack
  when:
    - ansible_kernel.split('.')[0:3] | join('.')  < '4.19'
    - kube_proxy_mode == 'ipvs'
  tags:
    - kube-proxy

- name: Modprobe nf_conntrack for kernels >= 4.19
  modprobe:
    name: nf_conntrack
    state: present
  when:
    - ansible_kernel.split('.')[0:3] | join('.')  >= '4.19'
    - kube_proxy_mode == 'ipvs'
  tags:
    - kube-proxy

- name: Persist ip_vs modules
  copy:
```

```
dest: /etc/modules-load.d/kube_proxy-ipvs.conf
content: |
  ip_vs
  ip_vs_rr
  ip_vs_wrr
  ip_vs_sh
  {% if enable_nf_conntrack is failed -%}
  nf_conntrack
  {%-   else -%}
  nf_conntrack_ipv4
  {%-   endif -%}
 when: kube_proxy_mode == 'ipvs'
```

2) При развёртывании kubespray появляется ошибка в Validate mirrors и the output has been hidden:

1. в файле roles/download/task/download_file.yml в name: download_file | Validate mirrors в поле no_log поставить "false" и перезапустить
2. по результатам логов исправить ошибку:
    1. в случае проблем с версиями calico удалить строки:

roles/download/default/main.yml

```
calicoctl_alternate_download_url:
"https://github.com/projectcalico/calicoctl/releases/download/{{
calico_ctl_version }}/calicoctl-linux-{{ image_arch }}"
```

roles/download/default/main.yml

```
 mirrors:
    - "{{ calicoctl_alternate_download_url }}"
    - "{{ calicoctl_download_url }}"
```

3) После развёртывания кластера узлы друг друга не видят, попробовать изменить настройки calico:

inventory/mycluster/group_vars/k8s_cluster/k8s-net-calico.yml

```
# Set calico network backend: "bird", "vxlan" or "none"
# bird enable BGP routing, required for ipip and no encapsulation modes
calico_network_backend: bird

# IP in IP and VXLAN is mutualy exclusive modes.
# set IP in IP encapsulation mode: "Always", "CrossSubnet", "Never"
calico_ipip_mode: 'Always'

# set VXLAN encapsulation mode: "Always", "CrossSubnet", "Never"
calico_vxlan_mode: 'Never'
```

4) Во время развёртывания происходит ошибка с сообщением "Timeout (12s) waiting for privilege escalation prompt":

1. Неправильные настройки sudo (должно выполняться без запроса пароля) – исправление см п.1.1 пп. 0
2. Если ошибка всё равно происходит необходимо добавить в файл ansible.cfg (в папку kubespray)
3. [defaults]
   timeout = 60

3 Структура программы

Программу условно можно разделить на две (зависимые) части, которые нуждаются в разных по составу и объему ресурсах (необходимые ресурсы описаны независимо в каждом из пунктов):

1) Аналитическая часть
2) Часть обработки данных

3.1 Аналитическая часть

К компонентам аналитической части относятся:

- база данных;
- подсистема авторизации/аутентификации;
- серверная часть аналитической системы;
- поисковой движок;
- подсистема аудита;
- файловое хранилище;
- серверная часть подсистемы сбора.

Особенности:

- большой объем необходимого дискового пространства,
- зависимость скорости работы от скорости дисков,
- ускорение работы за счет оперативной памяти (кеширование индексов и пр.),
- наличие моментов пикового потребления оперативной памяти (генерация отчетов, параллельная работа с возросшим числом пользователей и пр.).

Оптимально для работы наличие 100 Тб дискового пространства. Наличие 1 Тб ОЗУ оптимально для ускорения времени работы и отклика системы за счет кеширования.

Минимальным необходимым объемом ОЗУ для этой части программы является 256 Гб. Процессор, аналогичный Intel Xeon Gold 6144, 3,5 ГГц, 8 ядер в количестве 2 штук (или аналогичное количество потоков).

3.2 Часть обработки данных (Talisman.Поток)

К компонентам части обработки данных относятся:

- контроллер, управляющий конвейером обработки данных;
- Docreader;
- компоненты анализа текста;
- сборщики данных и т.д.

Особенности сервисов:

- Контроллер способен масштабировать количество обработчиков для балансировки скорости работы конвейера, как следствие, потребление памяти сервисами меняется во времени. При высоком темпе поступления данных из подсистемы сбора/внутренних источников потребление ресурсов может непредсказуемо возрастать, влияя на скорость обработки, поэтому 1 Тб ОЗУ являются оптимальными, 256 ГБ ОЗУ являются минимально необходимыми.
- Сервисы Docreader и анализа текстов могут использовать GPU для ускорения работы и обучения. С учетом необходимости дообучения моделей, используемых для анализа текста, на основе обратных связей и высокой эффективности вычислений для нейронных сверточных сетей, используемых в Docreader, оптимальным является наличие нескольких (от 2) видеокарт, аналогичных Nvidia Tesla T4 (по объему памяти). Процессор, аналогичный Intel Xeon Gold 6144, 3,5 ГГц, 8 ядер в количестве 2 штук (или аналогичное количество потоков). В данной части необходимо хранение скачанных данных, ожидающих обработки, моделей машинного обучения и пр. Постоянное хранение большого объема данных и быстрый доступ к ним не являются необходимым, поэтому требуется от 5 Тб дискового пространства.

Итого (суммарно):

- от 512 Гб до 2 Тб ОЗУ,
- (желательно) от 2 видеокарт, аналогичных Nvidia Tesla T4,
- Intel Xeon Gold 6144, 3,5 ГГц, 8 ядер (или аналогичный) в количестве 4 штук (или аналогичное количество потоков),

- от 105 Тб дискового пространства на контур.

3.3 Общая программная архитектура

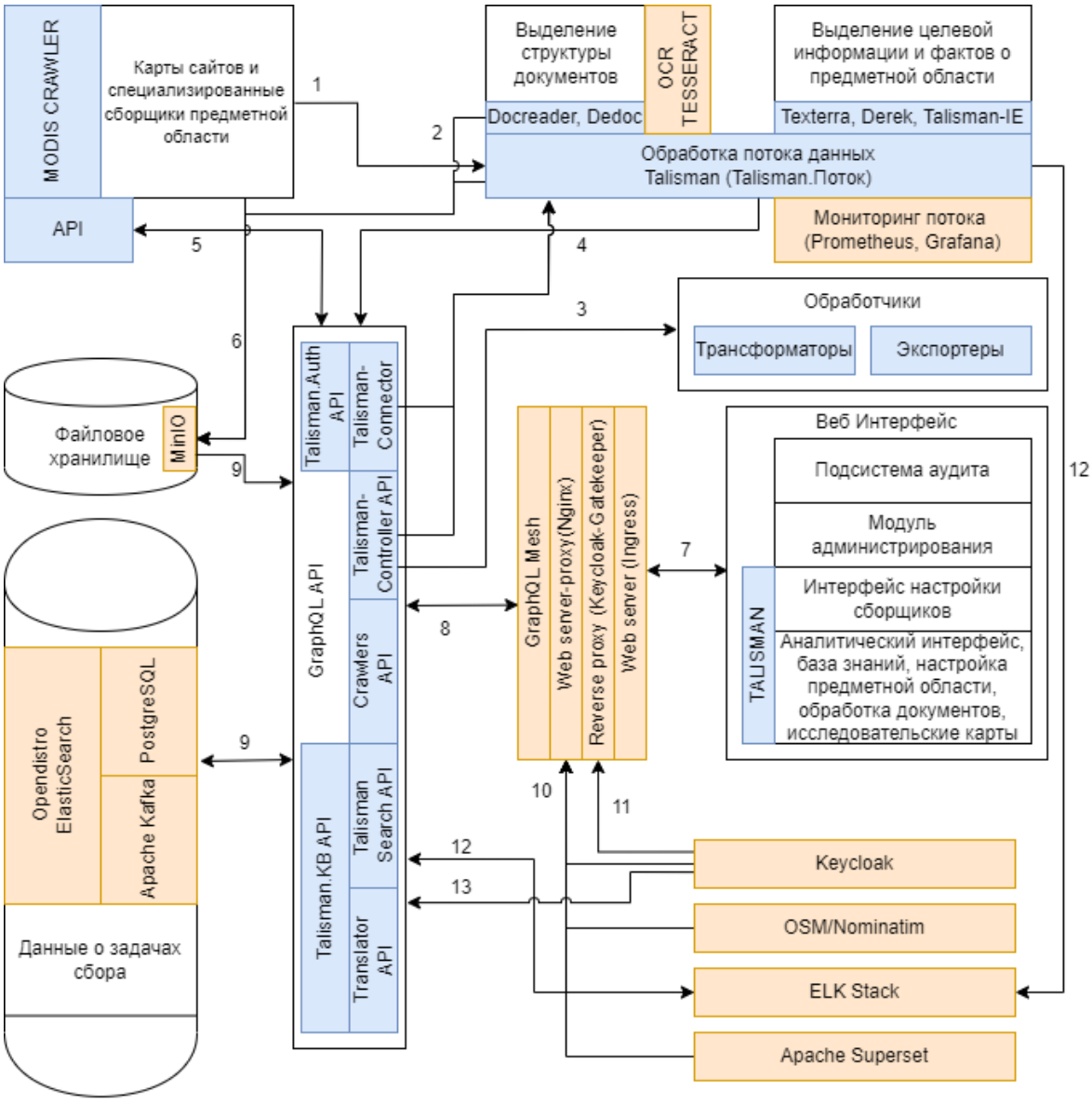

Рисунок 1 - Общая программная архитектура

Описание программной архитектуры:

1. Данные поступают в открытый контур с помощью сборщиков данных и передаются в очередь на обработку. Сборщики данных реализованы на инфраструктуре сбора данных MODIS CRAWLER, разработанной в ИСП РАН. Сборщиками можно также

управлять, передавая им задания на поиск и новые карты. Управление сборщиками осуществляется через программный интерфейс API.

2. Данные из подсистемы сбора передаются в подсистему обработки потока данных, реализованной на основе технологии Talisman.Поток, разработанной в ИСП РАН. Описание архитектуры и принципов работы системы Talisman.Поток приведено в следующем разделе.
3. Обработанные данные передаются через очередь в систему загрузки, приводятся к виду, в котором они хранятся в реестре документов и базе знаний и сохраняются.
4. Для добавления новых карт обхода и получения модели подсказок используется Crawlers.API.
5. Собранные файлы, импортированные файлы, и файлы, которые были рекурсивно извлечены из других файлов сохраняются в файловом хранилище MinIO. Файловое хранилище проксируется Talisman.KB API.
6. Управление работой сборщиков данных, работа по обогащению базы знаний, администрирование, работа с подсистемой аудита и подсистемой ведения пользователей ведется через веб-интерфейс.
7. Интерфейс Talisman требует наличие одной точки входа, поэтому несколько Graphql API, предоставляемых Talisman.KB, Crawlers, Translator, TSearch, TController, TConnector, объединяются в одно с помощью Mesh. Полученная одна точка входа проксируется nginx.
8. KB управляет извлечением и сохранением в реестр документов и базу знаний. Для быстрого поиска по данным строится индекс в ElasticSearch. Также производится синхронизация индекса и данных. Crawlers API сохраняет информацию о запусках, карты обхода и прочую информацию, необходимую для сбора в БД. Данные об оригинальных файлах и изображениях получаются из файлового хранилища.
9. Интерфейсы, предоставляемые Kibana, Keycloak и Nominatim проксируются с помощью nginx.
10. Для предоставления доступа только авторизированным пользователям используется прокси Keycloak-Gatekeeper. которое использует Keycloak для авторизации и аутентификации.
11. Журналы, которые пишутся сервисами, собираются с помощью Logstash, индексируются в elasticsearch и визуализируются в Kibana.

12. Для получения информации о пользователе, совершившем действия в системе Talisman.KB использует Keycloak как хранилище учетных данных пользователей

Программа состоит из следующих компонентов:

- Docreader, Dedoc – отвечают за преобразование входных файлов в произвольном формате в единый выходной формат. В ходе работы производится выделение текста (в том числе с помощью технологий оптического распознавания символов), выделение таблиц, выделение структуры документа, выделение и обработка вложенных файлов. Результатом работы является дерево логической структуры документа. Разработаны с использованием языка python 3.5.
- Tesseract – свободное ПО для распознавания текстов. Используется Docreader для извлечения текста из изображений и pdf без текстового слоя.
- Talisman-IE (TIE) – отвечает за лингвистический анализ текстовых документов: определение языка, сегментацию текстов, извлечение именованных сущностей, выделение бинарных семантических отношений, разрешение лексической многозначности. Реализован с использованием языка python 3.6.
- Talisman.Поток (Talisman-Controller) – система управления обработкой потока данных. Читает поток данных из СУБД Postgresql и в зависимости от очереди сообщения обрабатываются по настраиваемым потокам обработки, включающими последовательность сервисов-обогатителей. За управление потоками обработки отвечает подсистема Контроллер, который общается с сервисами-обогатителями с помощью их REST API и модифицирует оригинальное сообщение. Итоговое сообщение передается в Talisman.KB для сохранения. Написан на языке Scala.
- MinIO – S3 совместимое объектное хранилище. Используется для хранения неструктурированных данных – оригиналов документов, вложений и пр.
- PostgreSQL – свободная объектно-реляционная система управления базами данных. Используется для хранения ПБЗ, обработанных документов БДХ, данных сбора.
- Opendistro Elasticsearch – реализация Elasticsearch и плагинов к нему с открытым исходным кодом под лицензией Apache 2.0. Используется для индексации данных ПБЗ и БДХ для последующего поиска.
- ELK Stack – Elasticsearch, Logstash, Kibana, используется реализация Open Distro, стек для сбора, индексации и визуализации журнальной информации.

Используется для сбора журнала от всех сервисов, а также хранения «сырых» данных сбора. Настроена следующая политика индексации: каждый день создается новый индекс (в который записывается информация текущего дня), предыдущий индекс становится доступен только на чтение. В случае если суммарный размер индексов превышает настраиваемый порог, производится удаление наиболее старых индексов.

- Keycloak – продукт с открытым кодом для реализации технологии единого входа с возможностью управления доступом. Используется для ведения учетных данных пользователей программы.
- Keycloak-gatekeeper – прокси, проверяющее, что все внешние запросы поступают от авторизированных пользователей. Для этого используется Keycloak.
- OSM/Nominatim – некоммерческий веб-картографический проект. Используется для визуализации карты мира, получения координат и прямого и обратного поиска по карте. Использует отдельный PostgreSQL для хранения данных о карте мира.
- GraphQL Mesh – сервис, объединяющий несколько GraphQL API (от KB и Crawlers) в одно. Необходим в связи с архитектурными особенностями фронтенда.
- Nginx – открытый веб-сервер. Используется для проксирования всех сервисов.
- Talisman-KB – система, реализующий логику взаимодействия с базой знаний и предоставляющая API для части веб-интерфейса, отвечающего за процесс обогащения базы знаний пользователем. Также проксирует MinIO из соображений безопасности, осуществляет загрузку данных, приходящих из подсистемы Talisman.Поток, формирует и принимает пакеты, синхронизующие контуры программы. Написан на языке Scala с использованием фреймворков Play, Slick, Sangria.
- Talisman-Search – сервис, реализующий интерфейс полнотекстового поиска с использованием специального языка запросов по собранным и сформированным данных (документы и концепты системы). Написан на языке Scala с использованием фреймворков Play, Slick, Sangria.
- Talisman-Auth – сервис, предназначенный для хранения информации о правах и ролях пользователей системы. Написан на языке Scala с использованием фреймворков Play.

- Talisman-Connector – сервис, предназначенный для унификации интерфейса взаимодействия внешних систем импорта данных. Написан на языке Scala с использованием фреймворков Play.
- Translator – сервис, реализующий интерфейс переводчика произвольного текста с произвольного языка на произвольный язык, может использовать в качестве бэкенда самостоятельную модель на нейронной сети, Яндекс-переводчик или Google-переводчик. Написан на языке Python.
- Веб-интерфейс. Написан на TypeScript с использованием фреймворков React, GoJS, Ant.
- Apache Superset - открытая BI-платформа для визуализации данных, интегрированная с базой знаний.