



**«Платформа автоматизации наземного обеспечения рейсов и автоматизации менеджмента авиасервисов. (ПАНОРАМА)»**

## **Руководство администратора по установке**



## Содержание

<b>1</b>	<b>ПРЕДВАРИТЕЛЬНЫЕ ТРЕБОВАНИЯ .....</b>	<b>3</b>
1.1	Конфигурация аппаратных средств.....	3
1.2	ПК, с которого происходит установка.....	3
1.3	Сервера .....	3
<b>2</b>	<b>ИНСТРУКЦИЯ ПО УСТАНОВКЕ.....</b>	<b>4</b>
2.1	Инструкция по установке окружения .....	4
2.1.1	Kubernetes (k3s) .....	5
2.1.2	Opensearch, fluentd, dashboard.....	7
2.1.3	Kafka .....	8
2.1.4	Zeebe.....	9
2.1.5	Keycloak .....	10
2.2	Установка сервисов платформы .....	11
2.2.1	Фронт и справочные системы .....	11
2.2.2	Бэк-сервисы .....	12
<b>3</b>	<b>СОПРОВОЖДЕНИЕ.....</b>	<b>14</b>
3.1	Обновление компонентов.....	14
3.2	Откат обновления .....	14
3.3	Создание БД и пользователей для нее .....	14
3.4	Создание секретов в Kubernetes .....	14

# 1 ПРЕДВАРИТЕЛЬНЫЕ ТРЕБОВАНИЯ

## 1.1 Конфигурация аппаратных средств

«Платформа автоматизации наземного обеспечения рейсов и автоматизации менеджмента авиасервисов. (ПО «ПАНОРАМА»)» предполагает развертывание в kubernetes. В его состав входят master и worker ноды. Для корректной работы следует резервировать оба типа нод. В нашем случае это схема из трёх master и трёх worker нод. Так же необходимо три сервера для размещения БД.

## 1.2 ПК, с которого происходит установка

Для комфортного разворачивания системы необходима рабочая станция с root-доступом, доступом по сети к серверам, на которых будет развернут kubernetes кластер. На данной машине должен быть установлен Astra Linux.

Так же требуются инструменты входящие в поставку ОС:

Ansible

Git

Nano

## 1.3 Сервера

Система развертывается на AstraLinux «Воронеж». Остальное программное обеспечение разворачивается в процессе установки kubernetes кластера или БД.

Следует убедиться, что с сервера CI системы есть сетевой доступ к серверам, на которых будет разворачиваться Система.

Перед установкой инфраструктуры следует разметить диски на разделы. Данные диски в последствии будут использоваться для передачи их соответствующим системным сервисам.



## 2 ИНСТРУКЦИЯ ПО УСТАНОВКЕ

### 2.1 Инструкция по установке окружения

В окружение системы входит следующее ПО:  
Kubernetes (Оркестратор контейнеров),  
Kafka (Потоковая система обмена данными),  
Opensearch (Система полнотекстового поиска),  
Dashboards (Сервис визуализации данных),  
Fluentd (Сервис сбора события компонентов системы),  
Zeebe (Сервис исполнения сценариев),  
Keycloak (Сервис авторизации и аутентификации)

Все скрипты для успешного развертывания данного ПО передаются и находятся в Gitlab репозиториях Заказчика.

Далее будет описана установка каждого отдельного модуля.



### 2.1.1 Kubernetes (k3s)

Основные команды для развертывания описаны в readme.md файле. Все команды являются yml файлами, для выполнения которых и необходим ansible.

Команды следует выполнять с ПК, описанного в пункте 1.2.

В составе репозитория следует обратить внимание на следующие файлы: inventory/panorama\*.ini – файл описания серверов, на которых будет развернут кластер.

Group\_vars/prod (или другая среда) – файл описания переменных, в которых указываются сетевые адреса и имена кластера, которые будут использованы для доступа к нему, а так же токен, с которым создастся кластер

Group\_vars/all:

cluster\_cidr, service\_cidr, cluster\_dns, systemd\_dir, k3s\_version – переменные согласно документации k3s.

github\_url, docker\_repo, nexus\_repo – адреса соответствующих ресурсов и репозиторияев.

Далее будут описаны особенности работы ролей, на которые следует обратить внимание:

prepare-os.yml –

role: os/sudo-users:

В данной роли описываются учетные записи, владельцы которых будут администрировать данную систему. В папку roles/os/sudo-users/files следует положить публичные ssh-ключи, сгенерированные для данных учетных записей – они будут добавлены на сервера для доступа по ключу.

role: os/ntp:

В зависимости от используемого дистрибутива следует обратить внимание на адрес конфига chrony. В текущей конфигурации он прописан по адресу /etc/chrony.conf

prepare-k3s.yml – без особенностей

install-k3s.yml

В рамках данного плейбука происходит добавление зеркал на основные репозитории, используемые при установке окружения, а так же сборке компонентов EFTRMS. Описано в файле roles/k3s/k3s/all/templates

Оставшаяся часть инфраструктуры разворачивается с применением пайплайнов Gitlab CI.

Для успешного выполнения пайплайнов необходимо завести глобальную переменную, содержащую конфигурационный файл kubernetes-кластера, переведенный в base64 кодировку.

Получить его можно, выполнив команду

```
kubect1 config view --flatten=true
```

На любой из мастер-нод и заменив там

```
server: https://127.0.0.1:6443
```

на корректный внешний ip мастер-серверов.

Для использования пайплайнов необходимы

- 1) Gitlab docker runner на ПК с доступом к серверам, где планируется развертывание инфраструктуры.
- 2) Квалификация, включающая в себя понимание того, как работают пайплайны Gitlab CI.

Пайплайн Gitlab CI описан в файле gitlab-ci.yml.

В верхней части обычно описаны необходимые переменные Variables

Далее описаны шаги пайплайна и их порядок

```
stages:
  - deploy
  - mirroring
```

Далее подробно описан каждый шаг (джоб)

```
kafka-deploy:
  stage: deploy
```

Так же для ряда ПО необходимо описать persistent volume и persistent volume claim на разделы, разбитые в рамках пункта 1.3.

Описываются следующим образом в ручном режиме:

Необходимо получить UUID раздела командой blkid. Пример ответа:

```
/dev/sdak1: UUID="1f00cceb-74b0-4d0b-a62b-7b5bd71a7e16" TYPE="xfs"
PARTUUID="f3dec594-c84d-44da-bd44-60102387c4ba"
```

Далее описываем persistent volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: *имя раздела*
spec:
  capacity:
    storage: *размер ресурса*
  local:
    path: /dev/disk/by-uuid/*UUID полученный ранее*
    fsType: xfs
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: *необходимый storage class. В случае EIP local-storage*
  volumeMode: Filesystem
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - *список серверов, с которых раздел доступен*
```

И persistent volume claim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: *имя ресурса PVC. Обычно строится из суффикса, характерного для ПО + имени PODа. Будет указано отдельно для каждого вида ПО*
  namespace: *namespace в котором располагается сервис*
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: *размер ресурса*
  volumeName: *имя раздела PV*
```



```
storageClassName: local-storage  
volumeMode: Filesystem
```

## 2.1.2 Opensearch, fluentd, dashboard

Opensearch, fluentd, dashboard составляют один стек по сборке, хранению и визуализации логов. Далее по тексту он будет обозначен как Opensearch.

Перед установкой Opensearch стека следует описать соответствующие PV и PVC. Имя ресурса PVC – `opensearch-clustername-podName-*`.

В репозитории следует указать переменную, в которой будет указан файл доступа к кластеру kubernetes

```
variables:  
  KUBE_CONFIG:
```

При запуске могут быть ошибки, связанные с `vm.max_map_count`. Следует установить значение

```
sysctl -w vm.max_map_count=262144
```

Логи, которые необходимо исключить из коллекционирования, прописываются в файле

`fluentd.yml` в секции `custom-filter-config`

Например:

```
<filter kubernetes.**>  
  @type grep  
  <and>  
    <exclude>  
      key $.kubernetes.container_name  
      pattern ^activemq$  
    </exclude>  
    <exclude>  
      key $.log  
      pattern ^WARN | Transport Connection to: tcp:\\\\\\b(?::(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)?:\d{1,5}\b failed: java.io.EOFException$  
    </exclude>  
  </and>  
</filter>
```

В остальном репозиторий готов к установке.

### 2.1.3 Kafka

Перед установкой kafka следует описать соответствующие PV и PVC. Имя ресурса PVC – data-podName -\*.

В рамках репозитория производится установка Kafka и Zookeeper (kafka-deploy), так же по необходимости настройка репликации между ЦОДами (replication-config).

В рамках пайплайна обозначены следующие переменные: NAMESPACE\_KAFKA – неймспейс, который создается для ПО

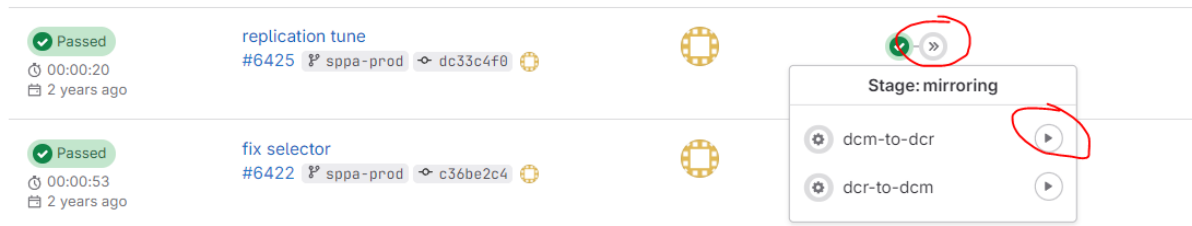
VIP\_\* - сетевой адрес, по которому будет доступно ПО снаружи kubernetes кластера.

Используется в файле k8s/service.yaml

```
externalIPs:
- __VIP__
```

Подменяется на соответствующий адрес в рамках пайплайна.

Шаг replication-config автоматически не выполняется, для его выполнения необходимо в ручном режиме запустить соответствующие шаги на вкладке build -> pipelines



Все необходимые для работы kafka topic в ПО Панорама также описываются в репозитории

В рамках пайплайна в \*-deploy секции указаны следующие переменные:

- KUBE\_CONFIG: переменная с конфигурационным файлом kubernetes кластера
- NAMESPACE\_NAME: неймспейс Кафки
- CLUSTER\_NAME: префикс среды
- RETENTION\_TIME: время сохранения сообщений в топиках
- RETENTION\_TIME\_SHORT: укороченное время сохранения сообщений для высоконагруженных топиков

В папке topics описаны сами топиками.

Для деплоя топиков в конкретную среду следует обратить внимание на секцию only:

```
variables:
```

В ней явно указано, какое коммит-сообщение следует оставлять для соответствующего деплоя.



## 2.1.4 Zeebe

Состоит из трех основных компонентов – worker, gateway и connector ноды.

Для успешного развертывания Zeebe следует изначально подготовить PV и PVC под воркер-ноды (см. раздел про Кубернетес).

Далее выполнить пайплайн. В самом пайплайне следует обратить внимание на файл

`extra-envs-config.yaml`, где описаны ресурсы, к которым Zeebe будет обращаться

`values_template.yaml` – описаны все переменные, с которыми будет запускаться Zeebe. Самые интересные для нас –

`zeebe.env`: описываются экспортеры, так же ниже описываются параметры PVC

`zeebe.initContainers`: контейнер для скачивания нужного экспортера

`zeebe.clusterSize`, `zeebe .partitionCount`, `zeebe .replicationFactor` –

настройка кластера воркеров

Далее это все применяется в шаблон `helm` с переменными, описанными в разделе соответствующей среды, например `dev-deploy`.

Важные тут –

`DISK_UUID_ZEEBE_*` – UUID для PVC

`CLUSTER_NAME` – имя кластера Zeebe

`ARTIFACTS_REPO` – адрес репозитория, от куда будет производиться загрузка исполняемых файлов экспортера

### 2.1.5 Keycloak

Состоит из трех компонентов – самого keycloak, oauth2-proxy, redis-базы.

Все файлы переменных находятся в подпапке deployment.

В файле keycloak следует обратить внимание на `initContainers` – место хранения актуального авторизационного провайдера.

В файле `oauth-proxy` важен раздел `containers.args` с ссылками на соответствующие адреса и конфиг

В `gitlab-ci.yml` следует обратить внимание на следующие переменные:

`DB_HOST`, `DB_PORT`, `DB_SECRET` – адрес, порт и пароль от бд

`HOSTNAME`, `IP`, `REALM` – имя, ip и имя рилма кейклока

`OAuth_CONF` – конфигурационный файл `oauth2`, закодированный через `base64`

Самое главное в данном `conf`. Файле – `issuerURL`, это адрес + рилм кейклока.

Должен соответствовать целевой системе.

Остальные настройки происходят уже непосредственно в рилме кейклока.

## 2.2 Установка сервисов платформы

Сервисы платформы делятся на две основные категории:

Фронт-сервисы

Бэк-сервисы

Настройка и установка внутри каждой группы схожа.

### 2.2.1 Фронт и справочные системы

Включают в себя следующие компоненты

`panorama-operational-anagement-ui-vue3`

`staff-mobile`

`planning-app`

`rostering-app`

`hub-ui-next`

Развертывание данных сервисов сводится к выполнению пайплайна с одним из трех условий:

1. Коммит в ветку `develop` развертывает сервис в `develop` окружении
2. Коммит с тегом `rc-*` развертывает сервис в `cert` окружении
3. Коммит с тегом `release-*` развертывает сервис в `prod` окружении

В самом пайплайне следует обратить внимание на следующие переменные:

`CI_PROJECT_TITLE` – при наличии - название проекта маленькими буквами, будет использовано, например, как имя `deployment-a` и `POD`

`ENDPOINT` – эндпоинт, по которому приложение будет ожидать к себе обращения

`IMAGE_NAME` – адрес, по которому будет выкладываться, и подхватываться образ.

`DOCKER_REGISTRY` – ссылка на докер-хранилище, в которое будет выкладываться образ.

`NEXUS_LOGIN` и `NEXUS_PASSWORD` – логин и пароль к докер-хранилищу.

Остальные важные настройки хранятся в файлах `.env.*` (в зависимости от среды) и `deployment.yml`

Файлы `.env.*` как правило уже готовы к установке. В случае необходимости следует правильно сопоставлять переменные между сервисами.

Например `VITE_API_PATH_BFF=/bff-planning-service` должна совпадать с настройкой в `appsettings.json` соответствующего сервиса бэка.

В `deployment.yml` построен на переменных и не требует изменений.

## 2.2.2 Бэк-сервисы

В бэк-сервисы входят следующие:

dmn  
 flightlegaggregator  
 hub-ui-next  
 linkedflightservice  
 planning-svc  
 planning-vocabs-prototype-service  
 rms-backforfront-service  
 rms-optaplanner-1-689569b9b4-skbtw  
 rms-optaplanner-65bc7f666-9k8hg  
 rms-read-model-operational-internal-service  
 rms-read-model-operational-service  
 rms-solvermanager-service  
 rms-traveltime-calc-service  
 rms-zeebe-worker  
 rms-zeebe-worker-sync  
 rostering-bff-service  
 vocabs-service  
 vocabs-hub  
 zeebe-kafka-groundconnections  
 zeebe-kafka-linked-flight-events-json  
 zeebe-kafka-netline-events-json  
 zeebe-kafka-notifications  
 zeebe-kafka-readmodelleg-events-json  
 zeebe-kafka-shifts-break  
 zeebe-kafka-vocabs-events-json  
 hub-api  
 hub-core  
 hub-notify  
 hub-pcm  
 hub-rb  
 hub-rmg  
 hub-sa

Все имеют схожую структуру за исключением случаев, отмеченных отдельно.

Настройки хранятся в `k8s/*/appsettings*.json`

Следует обратить внимание на секции `Zeebe`, `Authorize`, `Kafka`, `RoutePrefix`, `DatabaseConnection`, `EndpointPrefix` и заполнять их в соответствии с настройками конкретных сервисов.

В случае сервисов из проекта `resource-management-service` следует помнить, что все они собираются параллельно, за счет описания в переменных в `gitlab-ci.yml`.

Основные переменные в `gitlab-ci.yml`, которые могут использоваться в разных проектах:

`CI_PROJECT_TITLE` - имя сервиса  
`IMAGE_NAME` - адрес, куда будет залит образ после сборки  
`DEPLOYMENT_FILE` - файл описания деплоймента



PUBLISH – файл, который будет применен для сборки  
FOLDER – папка для хранения артефактов пайплайна  
FROMFILE – ссылка на файл настроек в проекте  
CONFIGFILE – имя файла конфига для деплоя  
CONFIGFILE1 – оно же, используется в `rms-zeebe-worker-sync` для отделения от асинхронного  
DOCKERFILE – `imgae` файл, использующийся для сборки

Отдельно следует уточнить о группе `zeebe-kafka-*`.

Образ для них общий, собирается в проекте `zeebe-kafka-connector-source-service`, без настроек.

Файл настроек подкладывается уже в рамках шага деплоя каждого конкретного именного проекта.

Для сборки следует, как и для фронта выполнить:

1. Коммит в ветку `develop` разворачивает сервис в `develop` окружении
2. Коммит с тегом `rc-*` разворачивает сервис в `cert` окружении
3. Коммит с тегом `release-*` разворачивает сервис в `prod` окружении

Конфигурация находится в файле `processor.xml` в корне проекта. В конфигурации содержится описание настроек плагинов, например `timer.PostgresTimerInputNode`, `scripting.ScriptNode`, `ibmmq.IbmMqOutputNode`.



## 3 СОПРОВОЖДЕНИЕ

### 3.1 Обновление компонентов

Обновление инфраструктурных компонентов выполняется в соответствии с инструкцией соответствующего ПО. Обычно достаточно в gitlab проекте заменить соответствующие образы на новые, однако, каждый раз следует проверять, не требуется ли дополнительной подготовки. Также следует изучить раздел отката обновлений – нужно ли для него сохранить какие-либо данные.

Обновление доменов и ESB происходит путем коммита в проект с соответствующими параметрами (см. соответствующий пункт по установке доменов или ESB).

### 3.2 Откат обновления

Откат инфраструктурных компонентов выполняется в соответствии с инструкцией соответствующего ПО. Обычно достаточно в gitlab проекте заменить соответствующие образы на старые, однако каждый раз следует проверять, не требуется ли дополнительной подготовки.

Откат доменов и esb происходит командой

```
Kubectl rollout undo -n namespace deployment/название-приложения
```

### 3.3 Создание БД и пользователей для нее

Создание БД и пользователей для нее производится в ручном режиме через консоль `psql`.

### 3.4 Создание секретов в Kubernetes

Создание секретов с пользователями БД возможно командой

```
kubectl create secret generic secret-name -n namespace \
  --from-literal=testdata=test \
  --from-literal=testdata2='test'
```

Где `generic` – базовый тип секрета

`--from-literal` – обозначение, что в секрет передается текстовая информация. Есть и другие варианты передачи информации, см. документацию по `kubectl secrets`.

Обновление секретов происходит следующим образом:

Получаем `base64` переменную

```
echo -n 'encode_My_Password' | base64
```

открываем секрет на редактирование

```
kubectl edit secret my-secret -n namespace
```

вставляем полученное в шаге `echo` значение в необходимое место.