

Общество с ограниченной ответственностью «Гиперион»

ОГРН 1237700921083 ИНН 9731127087

Программное обеспечение Платформа «Гиперион»

Технические характеристики и установка ПО

2024 г.

1. Технологии и требования	2
1.1. Сервер	2
1.2. Клиент	3
2. Установка	4
2.1. Подготовка к установке дистрибутива	4
2.2. Развертывание Keycloak	5
2.3. Развертывание Gitlab	7
2.4. Развертывание OKD	9
2.5. Развертывание Sonarqube	13
2.6. Развертывание Nexus	21
2.7. Развертывание ArgoCD	22
3. Используемые библиотеки и компоненты	25

1. Технологии и требования

1.1. Сервер

Минимальные аппаратные требования к серверному оборудованию, необходимые для развертывания Платформы, указаны в Таблице 1.

Т а б л и ц а 1 – Требования к серверному оборудованию

Роль / Компонент	Операционная система	Кол-во ядер, шт.	Объем оперативной памяти, ГБ	Объем хранилища, ГБ	Скорость сетевого интерфейса, Гб/с	Кол-во
bootstrap	Fedora CoreOS	4	16	120	10	1
master	Fedora CoreOS	4	16	120	10	3
worker	Fedora CoreOS	8	64	120	10	3
storage	Fedora CoreOS	8	32	1000	10	3
Gitlab	РЕД ОС 7.3	4	8	200	1	1

Указанные сервера могут быть как физическими, так и виртуальными. Сервера должны находиться в пределах одной сети (L2 домене) и иметь доступ в сеть Интернет. Наличие доступа в сеть Интернет является обязательным требованием на этапе развертывания Платформы.

Требования к программному обеспечению, необходимому для корректного функционирования Платформы:

- ОС, поддерживающая язык программирования Java (РЕД ОС 7.3 и т.п.);
- Пакет инструментов разработки – Java 8 JDK;
- Архиваторы, поддерживающие один или несколько указанных форматов файлов: .zip, .gzip, .tar
- СУБД, поддерживающая интерфейс JDBC (PostgreSQL 14.1, MySQL 8.0 и т.п.);
- Платформа для разработки, развертывания и эксплуатации классических и контейнерных приложений – OKD 4.

1.2. Клиент

Для автоматизированных рабочих мест пользователей используется рабочая станция в составе технических средств, представленных на Рисунке 2.



Рисунок 2. Структурная схема технических средств АРМ

Источник бесперебойного питания в составе АРМ пользователя обеспечивает автономность от источников электропитания на время достаточное для безаварийного завершения работы. Кроме того, устройство должно защитить оборудование от перепадов напряжения.

Таблица 2 содержит требования к аппаратной части АРМ.

Т а б л и ц а 2 – Требования к аппаратной части АРМ

Параметр	Значение
ЦПУ	Intel Pentium IV 1.2 ГГц и выше
ОЗУ	не менее 1 Гб
НЖМД	не менее 1 Гб
Видеоадаптер, поддерживающий видеорежим	разрешение не менее 1024x768
Сетевой адаптер	1 x Ethernet 10Base-T
Скорость подключения к серверной части	2 Мбит/сек
ИБП	30 минут работы в режиме отключенного электропитания

Требования к программному обеспечению, необходимому для корректного взаимодействия с Платформой:

- ОС Windows 7 и выше, AstraLinux Special Edition 2.12.43 и выше и т.п.;
- Браузер – Chrome 95 и выше, Opera 72 и выше, Firefox 93 и выше, Edge 95 и выше и т.п.

2. Установка

2.1. Подготовка к установке дистрибутива

Перед установкой дистрибутива Платформы необходимо развернуть следующие компоненты:

- Keycloak (п. 3.1);
- GitLab (п. 3.2);
- OKD (п. 3.3);
- Sonarqube (п. 3.4);
- Nexus (п. 3.5);
- ArgoCD (п. 3.6);

После развертывания указанных выше компонентов необходимо установить из предоставленного дистрибутива следующие компоненты:

- Портал разработчика;
- Интеграционные сервисы для работы с Gitlab, Argo CD, OKD, Sonar;
- Модуль Аудит;

Для этого необходимо в OKD создать проект huregion с необходимыми квотами, в котором будут развернуты компоненты.

Из предоставленного дистрибутива необходимо загрузить следующие образы в Nexus:

- audit_back_1_0_9.tar (Модуль аудит);
- front_v0_1_01.tar (Портал разработчика);
- int_argocd_1_0_33.tar (Интеграционный сервис Argo CD);
- int_gitlab_1_0_88.tar (Интеграционный сервис Gitlab);
- int_openshift_1_0_124.tar (Интеграционный сервис OKD);
- int_sonar_1_0_34.tar (Интеграционный сервис Sonar);
- portal_back_1_2_144.tar (Сервис для портала разработчика);

Далее необходимо создать в компоненте Gitlab репозиторий portal-infra и загрузить в него скрипты развертывания, которые хранятся в дистрибутиве в папке “portal-infra”.

В компоненте ArgoCD необходимо создать приложение hyperion со следующими параметрами:

```
namespace: hyperion
repo url: https://git.host/portal-infra.git
path: overlays/prod
cluster: https://kubernetes.default.svc
```

2.2. Развертывание Keycloak

Список действий, выполняемых на сервере, предназначенном для развертывания Keycloak:

- создать файл /etc/systemd/system/keycloak.service с правами 0644 и содержимым:

```
[Unit]
Description=Keycloak Server
After=network.target

[Service]
Type=idle
Environment="JAVA_OPTS=-Xms1024m -Xmx20480m -XX:MaxPerm
Size=768m"

User=keycloak
Group=keycloak
ExecStart=/opt/keycloak/bin/standalone.sh -Djboss.bind.address=0.0.0.0 -
Djboss.http.port=8080 -Djboss.https.port=8443 -
Djava.security.egd=file:/dev/urandom
```

TimeoutStartSec=600

TimeoutStopSec=600

[Install]

WantedBy=multi-user.target

- выполнить следующие команды из командной строки:

```
dnf install -y java-1.8.0-openjdk-headless firewalld
```

```
firewall-offline-cmd --add-port=8080/tcp
```

```
firewall-offline-cmd --add-port=8443/tcp
```

```
systemctl enable --now firewalld.service
```

```
curl -L https://downloads.jboss.org/keycloak/${sso_version}/keycloak-11.0.2.tar.gz -o sso.tar.gz
```

```
tar xzf sso.tar.gz
```

```
mkdir /opt/keycloak
```

```
useradd -d /opt/keycloak -M keycloak
```

```
cp -a keycloak-*/ */opt/keycloak/
```

```
chown -R keycloak:keycloak /opt/keycloak
```

```
rm sso.tar.gz
```

```
semanage fcontext -a -t bin_t "/opt/keycloak/bin(/.*)?"
```

```
restorecon -r -v /opt/keycloak/bin
```

```
sudo -u keycloak /opt/keycloak/bin/add-user-keycloak.sh -r master -u admin
```

-p

```
${ssoadmin_password}
```

```
systemctl enable --now keycloak
```

```
sudo -u keycloak /opt/keycloak/bin/kcadm.sh create realms -s  
realm=hyperion -s
```

```
enabled=true --no-config --server http://localhost:8080/auth --realm master -  
-user
```

```
admin --password ${ssoadmin_password}
```

где `ssoadmin_password` – пароль администратора Keycloak.

- далее необходимо настроить федерацию с LDAP-сервером. На примере интеграции с FreeIPA необходимо указать следующие настройки:

Vendor: Other

UUID LDAP attribute: `ipauniqueid`

Connection URL: `ldap://ipa-int.dev.hprn.ml`

Users DN: `cn=users,cn=accounts,dc=ipa,dc=dev,dc=hprn,dc=ml`

Bind DN: `uid=keycloak,cn=users,cn=accounts,dc=ipa,dc=dev,dc=hprn,dc=ml`

- далее необходимо добавить клиенты для интеграции с OKD, указав следующие значения:

Client ID: `hyperion`

Client Protocol: `openid-connect`

Access Type: `confidential`

Standard Flow Enabled: `On`

Valid Redirect URIs:

`https://git.dev.hprn.ml/`

`https://oauth-openshift.apps.okd.dev.hprn.ml/`

`https://sonar.apps.okd.dev.hprn.ml/`

Указанные URL используются для демонстративных целей. При развертывании требуется актуализация до используемых на целевой площадке.

Более подробно с инструкцией по развертыванию и настройке Keycloak можно ознакомиться по адресу:

https://www.keycloak.org/docs/11.0/server_installation/index.html

2.3. Развертывание Gitlab

Список действий, выполняемых на сервере, предназначенном для развертывания GitLab:

- создать файл `/etc/gitlab/gitlab.rb` с правами `0644` и содержимым:
`external_url 'https://${git-domain}'`


```

gitlab_rails['time_zone'] = 'Europe/Moscow'
gitlab_rails['gitlab_email_from'] = '${git-email}'
gitlab_rails['gitlab_email_display_name'] = 'Гиперион'
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = {
  'main' => {
    'label' => 'LDAP',
    'host' => '${ldap-host}',
    'port' => ${ldap-port},
    'uid' => 'uid',
    'encryption' => 'plain',
    'verify_certificates' => false,
    'bind_dn' => '${bind-dn}',
    'password' => '${bind-password}',
    'base' => '${ldap-base}',
    'attributes' => {
      'username' => ['uid'],
      'email' => ['mail'],
      'name' => 'displayName',
      'first_name' => 'givenName',
      'last_name' => 'sn'
    }
  }
}
gitlab_rails['backup_path'] = '/var/opt/gitlab/backups/full'
gitlab_rails['backup_keep_time'] = 604800 # 604800s = 7d || 2592000s = 30d
unicorn['worker_processes'] = 2
nginx['client_max_body_size'] = '4096m'
nginx['listen_port'] = 80
nginx['listen_https'] = false
nginx['proxy_set_headers'] = {

```

```
"X-Forwarded-Proto" => "https",
"X-Forwarded-Ssl" => "on"
}
nginx['real_ip_header'] = 'X-Forwarded-For'
nginx['real_ip_recursive'] = 'on'
prometheus_monitoring['enable'] = true
```

где `git-domain` – доменное имя сервера Gitlab, `git-email` – электронный адрес, с которого сервер Gitlab будет рассылать уведомления, `ldap-host` – адрес LDAP-сервера, `ldap-port` – порт LDAP-сервера, `bind-dn` – ldap-имя учетной записи, `bind-password` – пароль учетной записи, `ldap-base` – базовый фильтр LDAP для поиска пользователей.

- выполнить следующие команды из командной строки:

```
dnf install -y postfix polycoreutils firewallld
firewall-cmd --permanent --add-service=http
firewall-cmd --permanent --add-service=https
systemctl reload firewallld
systemctl enable --now postfix
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-
ce/script.rpm.sh |
bash
```

```
GITLAB_ROOT_PASSWORD=${gitlab_password}
GITLAB_ROOT_EMAIL=${gitlab_email} dnf install -y gitlab-ce
```

где `gitlab_password` – пароль администратора, `gitlab_email` – электронный адрес администратора.

2.4. Развертывание OKD

Список действий, выполняемых на сервере, предназначенном для развертывания OKD:

- создать файл `/root/okd-install/install-config.yaml` с правами 0644 и содержимым:

apiVersion: v1

baseDomain: \${base-domain}

metadata:

name: okd

compute:

- hyperthreading: Enabled

name: worker

replicas: 0

controlPlane:

hyperthreading: Enabled

name: master

replicas: 3

networking:

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

networkType: OpenShiftSDN

serviceNetwork:

- 172.30.0.0/16

platform:

none: {}

fips: false

pullSecret: '{"auths":{"fake":{"auth": "bar"}}}'

sshKey: \${public-key}

где base-domain – базовый домен кластера OKD, public-key – открытый ssh-ключ для доступа к серверам кластера OKD.

- ● ВЫПОЛНИТЬ В КОМАНДНОЙ СТРОКЕ:

```
curl -L https://github.com/openshift/okd/releases/download/4.6.0-0.okd-2020-12-12-135354/openshift-client-linux-4.6.0-0.okd-2020-12-12-135354.tar.gz -o client.tar.gz
curl -L https://github.com/openshift/okd/releases/download/4.6.0-0.okd-2020-12-12-135354/openshift-install-linux-4.6.0-0.okd-2020-12-12-135354.tar.gz -o install.tar.gz
cat *.tar.gz | tar zxf - -i -C /root
rm *.tar.gz
cp /root/oc /usr/local/bin/
/root/openshift-install create manifests --dir=/root/okd-install/
sed -i 's/mastersSchedulable: true/mastersSchedulable: False/'
/root/okdinstall/manifests/cluster-scheduler-02-config.yml
/root/openshift-install create ignition-configs --dir=/root/okd-install/
cd /root/okd-install/ && /usr/libexec/platform-python -m http.server 8000 &
mkdir /root/.kube
cp /root/okd-install/auth/kubeconfig /root/.kube/config
export KUBECONFIG=/root/okd-install/auth/kubeconfig
/root/openshift-install wait-for bootstrap-complete --dir /root/okd-install/ --log-level
debug
source <(oc completion bash)
oc completion bash > /root/.kube/completion.bash.inc
printf "
# Kubectl shell completion
source '/root//.kube/completion.bash.inc'
```

```
" >> /root/.bash_profile  
/root/openshift-install wait-for install-complete --dir /root/okd-install/ --log-  
level
```

```
debug
```

- далее необходимо настроить интеграцию с Keycloak. Для этого нужно создать Custom Resource с описанием провайдера аутентификации:

```
apiVersion: config.openshift.io/v1
```

```
kind: OAuth
```

```
metadata:
```

```
name: cluster
```

```
spec:
```

```
identityProviders:
```

```
- mappingMethod: claim
```

```
name: openid
```

```
openID:
```

```
ca:
```

```
name: ca-config-map
```

```
claims:
```

```
email:
```

```
- email
```

```
name:
```

```
- name
```

```
clientID: hyperion
```

```
clientSecret:
```

```
name: keycloak
```

```
extraAuthorizeParameters:
```

```
include_granted_scopes: "true"
```

```
extraScopes: []
```

```
issuer: https://KEYCLOAK_HOST/auth/realms/hyperion
```

```
type: OpenID
```

где KEYCLOAK_HOST – адрес сервера Keycloak.

2.5. Развертывание Sonarqube

Для развертывания Sonarqube в OKD необходимо создать проект hyperion-sonarqube с необходимыми квотами, в котором создать следующие объекты:

- PVC для СУБД PostgreSQL:
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: postgresql
 namespace: hyperion-sonarqube
spec:
 accessModes:
 - ReadWriteOnce
 resources:
 requests:
 storage: 50Gi
 storageClassName: rook-ceph-block
 volumeMode: Filesystem
- PVC для Sonarqube:
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: sonar-conf
 namespace: hyperion-sonarqube
spec:
 accessModes:
 - ReadWriteOnce
 resources:
 requests:

storage: 1Gi
storageClassName: rook-ceph-block
volumeMode: Filesystem

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: sonar-data
namespace: hyperion-sonarqube
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 10Gi
storageClassName: rook-ceph-block
volumeMode: Filesystem

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: sonar-extensions
namespace: hyperion-sonarqube
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 1Gi
storageClassName: rook-ceph

```

-block
  volumeMode: Filesystem
---
  kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: sonar-logs
  namespace: hyperion-sonarqube
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: rook-ceph-block
  volumeMode: Filesystem
- Deployment СУБД Postgresql:
  kind: Deployment
  apiVersion: apps/v1
  metadata:
    name: postgresql
    namespace: hyperion-sonarqube
  spec:
    replicas: 1
    selector:
      matchLabels:
        name: postgresql
    template:
      metadata:
        labels:

```



```
name: postgresql
spec:
  volumes:
    - name: postgresql-data
  persistentVolumeClaim:
    claimName: postgresql
  containers:
    - resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 500m
          memory: 512Mi
      readinessProbe:
        exec:
          command:
            - /usr/libexec/check-container
            initialDelaySeconds: 5
            timeoutSeconds: 1
            periodSeconds: 10
            successThreshold: 1
            failureThreshold: 3
        name: postgresql
      livenessProbe:
        exec:
          command:
            - /usr/libexec/check-container
            '--live'
            initialDelaySeconds: 120
```

timeoutSeconds: 10
periodSeconds: 10
successThreshold: 1
failureThreshold: 3
env:
- name: POSTGRESQL_USER
valueFrom:
secretKeyRef:
name: postgresql
key: database-user
- name: POSTGRESQL_PASSWORD
valueFrom:
secretKeyRef:
name: postgresq
key: database-password
- name: POSTGRESQL_DATABASE
valueFrom:
secretKeyRef:
name: postgresql
key: database-name
ports:
- containerPort: 5432
protocol: TCP
imagePullPolicy: IfNotPresent
volumeMounts:
- name: postgresql-data
mountPath: /var/lib/pgsql/data
terminationMessagePolicy: File
image: 'docker.io/library/postgres:12'
restartPolicy: Always

```
strategy:
  type: Recreate
  revisionHistoryLimit: 2
- Deployment Sonarqube:
  kind: Deployment
  apiVersion: apps/v1
  metadata:
    name: sonarqube
    namespace: hyperion-sonarqube
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: sonarqube
    template:
      metadata:
        labels:
          app: sonarqube
      spec:
        volumes:
          - name: conf
            persistentVolumeClaim:
              claimName: sonar-conf
          - name: data
            persistentVolumeClaim:
              claimName: sonar-data
          - name: logs
            persistentVolumeClaim:
              claimName: sonar-logs
          - name: extensions
```

persistentVolumeClaim:
claimName: sonar-extensions
containers:
- resources:
limits:
cpu: '2'
memory: 8Gi
requests:
cpu: '2'
memory: 8Gi
name: sonarqube
env:
- name: SONARQUBE_JDBC_URL
value: 'jdbc:postgresql://postgresql/sonarqube'
- name: SONARQUBE_JDBC_USERNAME
valueFrom:
secretKeyRef:
name: postgresql
key: database-user
- name: SONARQUBE_JDBC_PASSWORD
valueFrom:
secretKeyRef:
name: postgresql
key: database-password
ports:
- containerPort: 9000
protocol: TCP
imagePullPolicy: Always
volumeMounts:
- name: conf

```
mountPath: opt/sonarqube/conf
- name: data
mountPath: opt/sonarqube/data
- name: logs
mountPath: opt/sonarqube/logs
- name: extensions
mountPath: opt/sonarqube/extensions
image: 'docker.io/porscheinformatik/sonarqube:8'
restartPolicy: Always
strategy:
type: Recreate
revisionHistoryLimit: 2
```

Далее необходимо произвести следующие шаги:

- на странице https://SONAR_HOST/admin/marketplace необходимо установить плагин "OpenID Connect Authentication for SonarQube" и перезапустить сервер Sonarqube;
- далее необходимо перейти в основные настройки https://SONAR_HOST/admin/settings и указать следующие значения:
sonar.core.serverBaseUrl = [https:// SONAR_HOST](https://SONAR_HOST)
- далее необходимо перейти в настройки аутентификации https://SONAR_HOST/admin/settings?category=security и настроить интеграцию с keycloak:
sonar.auth.oidc.enabled = true
sonar.auth.oidc.issuerUri =
https://KEYCLOAK_HOST/auth/realms/hyperion

21

```
sonar.auth.oidc.clientId.secured = hyperion
sonar.auth.oidc.clientSecret.secured = SECRET
sonar.auth.oidc.scopes = openid email profile
sonar.auth.oidc.allowUsersToSignUp = true
```

20

sonar.auth.oidc.groupsSync = true

sonar.auth.oidc.loginButtonText = Гиперион

где SECRET - секрет клиента hyperion в Keycloak, KEYCLOAK_HOST – адрес сервера Keycloak, SONAR_HOST – адрес сервера Sonarqube.

2.6. Развертывание Nexus

Для развертывания Nexus в OKD необходимо добавить соответствующий оператор Nexus Operator. Далее необходимо создать проект hyperion-nexus с необходимыми квотами, в который добавить экземпляр Nexus следующего вида:

apiVersion: apps.m88i.io/v1alpha1

kind: Nexus

metadata:

name: nexus3

namespace: hyperion-nexus

spec:

serviceAccountName: nexus3

resources:

limits:

cpu: '2'

memory: 2Gi

requests:

cpu: '2'

memory: 2Gi

readinessProbe:

failureThreshold: 3

initialDelaySeconds: 240

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 15

livenessProbe:

failureThreshold: 3
initialDelaySeconds: 240
periodSeconds: 10
successThreshold: 1
timeoutSeconds: 15
serverOperations: {}
imagePullPolicy: Always
automaticUpdate:
disabled: true
minorVersion: 28
networking:
exposeAs: Route
tls: {}
image: docker.io/sonatype/nexus:3
replicas: 1
persistence:
persistent: true
storageClass: rook-ceph-block
volumeSize: 100Gi

2.7. Развертывание ArgoCD

Для развертывания ArgoCD в OKD необходимо добавить соответствующий оператор ArgoCD Operator.

Далее необходимо создать проект hyperion-argocd с необходимыми квотами, в который добавить экземпляр ArgoCD следующего вида:

apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
name: argocd

```
namespace: hyperion-argocd
spec:
  kustomizeBuildOptions: --load_restrictor none
  oidcConfig: |
  name: Keycloak
  issuer: https://keycloak.host/auth/realms/hyperion
  clientID: argocd
  clientSecret: clientSecret
  requestedScopes: ["openid", "profile", "email", "groups"]
  applicationInstanceLabelKey: argocd.argoproj.io/instance
  dex:
    openShiftOAuth: true
    grafana:
      enabled: true
    prometheus:
      enabled: true
    rbac:
      policy: |
        g, system:authenticated, role:admin
        g, root, role:admin
        g, ArgoCDAdmins, role:admin
    repositoryCredentials: |
      - url: https://git.host
    passwordSecret:
      name: git-secret
      key: password
    usernameSecret:
      name: git-secret
      key: username
  server:
```


host: argocd.host

insecure: true

route:

enabled: true

3. Используемые библиотеки и компоненты

Портал разработчика – единый интерфейс для работы с платформой.

Модуль Аудит – модуль обеспечивающий логирование событий, которые происходят на Платформе Гиперион.

Контейнерная платформа (OKD/OpenShift) – система для разработки и выполнения приложений, упакованных в контейнеры.

Хранилище данных (Ceph Storage) – определяемые программным обеспечением унифицированные решения для хранения данных.

Управление доступом (FreeIPA) – управление доступом субъектов к объектам на основе списков управления доступом или матрицы доступа.

Хранилище исходного кода (GitLab) – место, где хранятся и поддерживаются какие-либо данные.

Хранение артефактов сборки (Sonatype Nexus) – платформа, с помощью которой разработчики могут проксировать, хранить и управлять зависимостями, образами, а также распространять свое программное обеспечение.

Argo CD – это декларативный инструмент непрерывной доставки для Kubernetes.

Управление пользовательскими учетными данными (Keycloak) – комплекс подходов, практик, технологий и специальных программных средств для управления учетными данными пользователей, системами контроля и управления доступом, позволяющий повысить безопасность и производительность информационных систем при одновременном снижении затрат, оптимизации времени простоя и сокращения количества повторяющихся задач.

Мониторинг (Prometheus) – процесс централизованного сбора метрик работоспособности всех компонентов системы. Используется для получения картины работы как системы в целом, так и отдельных ее компонентов.

Сбор логов (Elasticsearch) – процесс централизованного сбора журналов и логов всех компонентов системы для дальнейшего анализа.

Интеграционные сервисы для работы с Gitlab, Argo CD, OKD, Sonar – это набор сервисов, которые обеспечивают работоспособность Портала разработчика и отвечают за интеграцию с соответствующими инфраструктурными компонентами.