

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Симулятор моделирования PVT-свойств
флюидов "Равновесие"

Руководство администратора
Установка и эксплуатация в Kubernetes

Наименование ПО	Симулятор моделирования PVT-свойств флюидов "Равновесие"
Тип документа	Административное руководство
Версия	1.0
Дата	28 августа 2026 г.
Статус	Для подготовки развёртывания

Документ содержит требования, порядок установки, эксплуатационные процедуры и шаблоны конфигурации.

Оглавление

Оглавление	2
1. Назначение документа	4
2. Состав приложения	4
3. Важные ограничения текущей версии	4
4. Требования к инфраструктуре	5
5. Подготовка исходников перед сборкой	5
5.1. Frontend	5
5.2. Backend и EOS	6
6. Сборка и публикация образов	6
7. Kubernetes-манифесты	7
7.1. Namespace, Secret и PVC	7
7.2. Backend	7
7.3. Frontend	9
7.4. Ingress	9
8. Порядок установки	10
9. Проверка после установки	11
9.1. Состояние Kubernetes	11
9.2. API изнутри кластера	11
9.3. Внешняя проверка	11
10. Эксплуатация	11
10.1. Просмотр логов	11
10.2. Обновление	12
10.3. Откат	12
10.4. Ротация API-ключа	12
10.5. Очистка кэша модели	12
10.6. Акустические модели	12
11. Резервное копирование и восстановление	12
12. Диагностика	13
Frontend возвращает 404 при обновлении страницы	13
Запросы /api/* возвращают 404	13

API возвращает 401	13
Backend pod не стартует	13
PVC не подключается при обновлении	13
Проекты теряются после перезапуска	13
predict_ready равен false	13
 13. Рекомендуемые доработки перед production	 13

1. Назначение документа

Документ описывает развёртывание ПО Симулятор моделирования PVT-свойств флюидов "Равновесие" в Kubernetes для следующего контура:

- веб-интерфейс React/Vite;
- FastAPI API;
- математическая EOS-модель advanced-pvt-model версии v2.0.0 или новее;
- файловое хранилище проектов;
- при необходимости — акустическая модель расчёта дебита.

2. Состав приложения

Компонент	Путь	Назначение	Порт
Frontend	frontend-main/	React/Vite SPA	80 в контейнере
Backend	backend-main/	FastAPI, проекты, расчёты, EOS, акустика	8000
EOS	advanced-pvt-model-develop/v2.0.0/	Cython-модули PR/SRK и pvteos	встроен в backend-образ
Project storage	PVC	JSON, загруженные файлы, кэш расчётов, модели	/data/projects

Рекомендуемая маршрутизация:

Пример: text

```
https://pvt.example.ru/ -> frontend Service:80
https://pvt.example.ru/api/* -> backend Service:8000, префикс /api удаляется Ingress-контроллером
```

В FastAPI задан `root_path="/api"`, однако обработчики объявлены как `/projects`, `/componentsOptions` и т. п. Поэтому Ingress должен передавать бэкэнду путь без внешнего префикса `/api`.

3. Важные ограничения текущей версии

Перед промышленным развёртыванием необходимо учитывать следующее.

1. Backend должен запускаться в одной реплике. Данные хранятся на файловом томе, а блокировки дерева проектов и кэша находятся в памяти одного процесса. Несколько pod могут одновременно изменить один файл и повредить данные.
2. Для Deployment backend рекомендуется стратегия Recreate. Она исключает одновременный доступ старого и нового pod к PVC во время обновления.
3. frontend-main/Dockerfile только собирает /app/dist, но не содержит runtime-стадию с Nginx и не запускает сервер. Ниже приведён рекомендуемый вариант.
4. Frontend использует zod, но пакет отсутствует в прямых dependencies файла package.json. Перед сборкой следует добавить совместимую версию и обновить lock-файл.
5. VITE_TEMP_PASS встраивается в JavaScript-бандл и одновременно используется как API-ключ. Пользователь браузера может увидеть его в коде приложения. Это не полноценная система аутентификации. Для закрытого контура допустимо только как временная мера; для промышленной эксплуатации нужен OIDC/OAuth2 или авторизация на reverse proxy без секрета во frontend.
6. После успешного ввода пароля форма входа всё равно вызывает setIsError(true). Переход обычно происходит благодаря изменению состояния, но логику следует исправить до релиза.

7. В приложении нет отдельного `/healthz` или `/readyz`. В примерах используются TCP-пробы. Они подтверждают, что порт открыт, но не проверяют EOS и доступность PVC.
8. Backend-образ с EOS и CPU PyTorch будет крупным. Следует заранее проверить лимиты registry, скорость скачивания образа и свободное место на Kubernetes-узлах.

4. Требования к инфраструктуре

- Kubernetes 1.29 или новее;
- установленный Ingress Controller, в примерах — `ingress-nginx`;
- StorageClass с поддержкой `ReadWriteOnce`;
- приватный OCI registry, доступный узлам кластера;
- DNS-имя приложения;
- TLS-сертификат или `cert-manager`;
- Linux amd64 nodes, если Cython-модули собираются для amd64;
- минимум для первого стенда:
- backend: request 2 CPU / 4 GiB RAM, limit 8 CPU / 12 GiB RAM;
- frontend: request 50m CPU / 64 MiB RAM;
- PVC: от 20 GiB, больше при работе с WAV и акустическими моделями.

Ресурсы являются стартовой оценкой. После нагрузочного теста их нужно скорректировать по фактическому CPU, памяти, времени EOS-расчётов и размеру проектов.

5. Подготовка исходников перед сборкой

5.1. Frontend

Добавить `zod` как прямую production-зависимость и зафиксировать lock-файл:

Пример: `bash`

```
cd frontend-main
npm install zod
npm run build
```

Исправить обработчик входа так, чтобы ошибка устанавливалась только при неверном пароле.

Заменить `frontend-main/Dockerfile` на многостадийный образ:

Пример: `dockerfile`

```
FROM node:20.19.0-alpine AS builder

ARG VITE_TEMP_PASS
ARG VITE_PVTSIM_BASE_URL
ENV VITE_TEMP_PASS=$VITE_TEMP_PASS
ENV VITE_PVTSIM_BASE_URL=$VITE_PVTSIM_BASE_URL

WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:1.27-alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 80
```

Файл `frontend-main/nginx.conf`:

Пример: nginx

```
server {
    listen 80;
    server_name _;

    root /usr/share/nginx/html;
    index index.html;

    location = /healthz {
        access_log off;
        add_header Content-Type text/plain;
        return 200 "ok\n";
    }

    location / {
        try_files $uri $uri/ /index.html;
    }

    location ~* \.(js|css|png|jpg|jpeg|gif|svg|ico|woff|woff2)$ {
        expires 7d;
        add_header Cache-Control "public, immutable";
        try_files $uri =404;
    }
}
```

VITE_PVTSIM_BASE_URL должен быть внешним origin без /api, например https://pvt.example.ru. Значение VITE_TEMP_PASS на текущем этапе должно совпадать с PVT_API_KEY backend.

5.2. Backend и EOS

Для рабочей матмодели нужен Dockerfile.eos. Он ожидает такой build context:

Пример: text

```
build-context/
backend/
advanced-pvt-model/
```

Текущие локальные каталоги называются backend-main и advanced-pvt-model-develop, поэтому нельзя просто выполнить docker build из корня без подготовки контекста. GitLab CI уже создаёт временный контекст и копирует backend в backend/.

Локальный эквивалент на Linux:

Пример: bash

```
BUILD_CONTEXT="$(mktemp -d)"
cp -a backend-main "$BUILD_CONTEXT/backend"
cp -a advanced-pvt-model-develop "$BUILD_CONTEXT/advanced-pvt-model"
docker build \
  -f "$BUILD_CONTEXT/backend/Dockerfile.eos" \
  -t registry.example.ru/pvt/backend:0.1.0 \
  "$BUILD_CONTEXT"
```

После сборки Dockerfile.eos проверяет импорт pvteos, pr, srk, наличие новых CVD-полей и импорт CPU PyTorch. Сборка должна завершиться без пропуска этих проверок.

6. Сборка и публикация образов

Используйте неизменяемые теги: номер версии или digest. Тег latest неудобен для аудита и отката.

Пример: bash

```
docker build \
  --build-arg VITE_PVTSIM_BASE_URL=https://pvt.example.ru \
  --build-arg VITE_TEMP_PASS="$PVT_API_KEY" \
  -t registry.example.ru/pvt/frontend:0.1.0 \
  frontend-main

docker push registry.example.ru/pvt/frontend:0.1.0
docker push registry.example.ru/pvt/backend:0.1.0
```

Не печатайте значение API-ключа в CI-логах. Учтите, что текущая frontend-сборка всё равно помещает его в публичный JS-бандл.

7. Kubernetes-манифесты

Ниже приведён минимальный рабочий шаблон. Замените домен, StorageClass, registry и теги образов.

7.1. Namespace, Secret и PVC

Не храните реальное значение секрета в Git. Используйте External Secrets, Sealed Secrets, Vault либо создайте Secret непосредственно в кластере.

Пример: yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: pvt
---
apiVersion: v1
kind: Secret
metadata:
  name: pvt-secrets
  namespace: pvt
type: Opaque
stringData:
  PVT_API_KEY: "${PVT_API_KEY}"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvt-projects
  namespace: pvt
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard-rwo
  resources:
    requests:
      storage: 20Gi
```

Если registry приватный:

Пример: bash

```
kubectl -n pvt create secret docker-registry registry-credentials \
  --docker-server=registry.example.ru \
  --docker-username="${REGISTRY_USER}" \
  --docker-password="${REGISTRY_PASSWORD}"
```

7.2. Backend

Пример: yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pvt-backend
  namespace: pvt
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: pvt-backend
  template:
    metadata:
      labels:
        app: pvt-backend
    spec:
      imagePullSecrets:
        - name: registry-credentials
      securityContext:
        fsGroup: 1000
      containers:
        - name: backend
          image: registry.example.ru/pvt/backend:0.1.0
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 8000
          env:
            - name: PROJECTS_DIR
              value: /data/projects
            - name: PVT_API_KEY
              valueFrom:
                secretKeyRef:
                  name: pvt-secrets
                  key: PVT_API_KEY
            - name: PVT_MODEL_XLSX_PATH
              value: /app/test.xlsx
            - name: PVT_ADAPTATION_ENABLED
              value: "1"
          volumeMounts:
            - name: projects
              mountPath: /data/projects
          startupProbe:
            tcpSocket:
              port: http
            failureThreshold: 60
            periodSeconds: 5
          readinessProbe:
            tcpSocket:
              port: http
            periodSeconds: 10
            failureThreshold: 3
          livenessProbe:
            tcpSocket:
              port: http
            initialDelaySeconds: 30
            periodSeconds: 20
            failureThreshold: 3
          resources:
            requests:
              cpu: "2"
              memory: 4Gi
            limits:
              cpu: "8"
              memory: 12Gi
          volumes:
            - name: projects
              persistentVolumeClaim:
                claimName: pvt-projects
---
apiVersion: v1
kind: Service
metadata:
  name: pvt-backend
  namespace: pvt
spec:
```

```
selector:
  app: pvt-backend
ports:
  - name: http
    port: 8000
    targetPort: http
```

7.3. Frontend

Пример: yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pvt-frontend
  namespace: pvt
spec:
  replicas: 2
  selector:
    matchLabels:
      app: pvt-frontend
  template:
    metadata:
      labels:
        app: pvt-frontend
    spec:
      imagePullSecrets:
        - name: registry-credentials
      containers:
        - name: frontend
          image: registry.example.ru/pvt/frontend:0.1.0
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 80
          readinessProbe:
            httpGet:
              path: /healthz
              port: http
              periodSeconds: 10
          livenessProbe:
            httpGet:
              path: /healthz
              port: http
              periodSeconds: 20
          resources:
            requests:
              cpu: 50m
              memory: 64Mi
            limits:
              cpu: 500m
              memory: 256Mi
---
apiVersion: v1
kind: Service
metadata:
  name: pvt-frontend
  namespace: pvt
spec:
  selector:
    app: pvt-frontend
  ports:
    - name: http
      port: 80
      targetPort: http
```

7.4. Ingress

Для корректного удаления /api удобно использовать отдельный Ingress backend.

Пример: yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pvt-backend
  namespace: pvt
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/proxy-body-size: 1024m
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "600"
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - pvt.example.ru
      secretName: pvt-tls
  rules:
    - host: pvt.example.ru
      http:
        paths:
          - path: /api(/|$)(.*)
            pathType: ImplementationSpecific
            backend:
              service:
                name: pvt-backend
                port:
                  number: 8000
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pvt-frontend
  namespace: pvt
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - pvt.example.ru
      secretName: pvt-tls
  rules:
    - host: pvt.example.ru
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: pvt-frontend
                port:
                  number: 80
```

Лимит 1024m нужен только для больших акустических записей. Если акустика не используется, уменьшите его, например до 64m.

8. Порядок установки

1. Исправить перечисленные блокирующие проблемы frontend и собрать оба образа.
2. Опубликовать образы с неизменяемыми тегами в registry.
3. Создать namespace и registry secret.
4. Создать pvt-secrets безопасным способом.
5. Создать PVC и убедиться, что он перешёл в Bound.
6. Применить Deployment и Service backend.
7. Дождаться готовности backend и проверить логи.
8. Применить Deployment и Service frontend.
9. Применить Ingress и TLS.

10. Выполнить функциональную проверку через внешний URL.

Пример команд:

Пример: bash

```
kubectl apply -f namespace-secret-pvc.yaml
kubectl -n pvt get pvc
kubectl apply -f backend.yaml
kubectl -n pvt rollout status deployment/pvt-backend --timeout=10m
kubectl apply -f frontend.yaml
kubectl -n pvt rollout status deployment/pvt-frontend --timeout=5m
kubectl apply -f ingress.yaml
```

9. Проверка после установки

9.1. Состояние Kubernetes

Пример: bash

```
kubectl -n pvt get pods,svc,ingress,pvc
kubectl -n pvt describe deployment pvt-backend
kubectl -n pvt logs deployment/pvt-backend --tail=200
```

9.2. API изнутри кластера

Пока отдельного health endpoint нет, можно проверить список проектов:

Пример: bash

```
kubectl -n pvt run api-check --rm -i --restart=Never \
  --image=curlimages/curl:8.12.1 -- \
  curl -fsS -H "Authorization: Bearer ${PVT_API_KEY}" \
  http://pvt-backend:8000/projects
```

9.3. Внешняя проверка

Пример: bash

```
curl -I https://pvt.example.ru/
curl -fsS \
  -H "Authorization: Bearer ${PVT_API_KEY}" \
  https://pvt.example.ru/api/projects
```

Проверить в браузере:

- открывается форма входа;
- после входа загружается список проектов;
- создаётся проект с ручным компонентным составом;
- выполняется один тестовый PR/SRK-расчёт;
- после перезапуска backend pod проект остаётся доступным.

Если используется акустическая модель:

Пример: bash

```
curl -fsS \
  -H "Authorization: Bearer ${PVT_API_KEY}" \
  https://pvt.example.ru/api/acoustic/status
```

Ожидается "predict_ready": true только при наличии PyTorch и обученной модели на PVC.

10. Эксплуатация

10.1. Просмотр логов

Пример: bash

```
kubectl -n pvt logs -f deployment/pvt-backend
kubectl -n pvt logs -f deployment/pvt-frontend
kubectl -n ingress-nginx logs deployment/ingress-nginx-controller --tail=200
```

В production рекомендуется собирать stdout/stderr через Loki, Elasticsearch/OpenSearch или другую централизованную систему.

10.2. Обновление

Пример: bash

```
kubectl -n pvt set image deployment/pvt-backend \
  backend=registry.example.ru/pvt/backend:0.1.1
kubectl -n pvt rollout status deployment/pvt-backend --timeout=10m

kubectl -n pvt set image deployment/pvt-frontend \
  frontend=registry.example.ru/pvt/frontend:0.1.1
kubectl -n pvt rollout status deployment/pvt-frontend --timeout=5m
```

Перед обновлением backend сделайте snapshot PVC или резервную копию. При несовместимом формате tree.json автоматической миграции базы данных нет.

10.3. Откат

Пример: bash

```
kubectl -n pvt rollout history deployment/pvt-backend
kubectl -n pvt rollout undo deployment/pvt-backend
kubectl -n pvt rollout undo deployment/pvt-frontend
```

Для backend предпочтительнее явно вернуть предыдущий image tag. Откат образа не откатывает данные на PVC.

10.4. Ротация API-ключа

В текущей реализации ротация требует одновременно:

1. изменить PVT_API_KEY в Kubernetes Secret;
2. пересобрать frontend с тем же новым значением VITE_TEMP_PASS;
3. обновить frontend и backend;
4. завершить проверку до удаления старого образа.

Это ещё одна причина заменить временную схему авторизации перед production.

10.5. Очистка кэша модели

Кэш находится в:

Пример: text

```
/data/projects/<project_id>/results/model_cache
```

Удаляйте только каталог конкретного проекта после резервного копирования и при остановленном backend pod. Не удаляйте корень /data/projects.

10.6. Акустические модели

Обученная модель должна находиться на PVC:

Пример: text

```
/data/projects/acoustic/_models/<model_id>/meta.json
/data/projects/acoustic/_models/<model_id>/model.pt
```

Для регулярного обновления моделей лучше применять initContainer или отдельный управляемый Job, который скачивает версию модели из объектного хранилища. Ручной kubectl ср допустим только для первичной проверки.

11. Резервное копирование и восстановление

Обязательный объект резервного копирования — PVC pvt-projects. Он содержит проекты, результаты, кэш и акустические модели.

Рекомендуемый вариант:

- CSI VolumeSnapshot по расписанию;
- отдельная копия в S3-совместимое хранилище;
- политика хранения daily/weekly/monthly;
- регулярная тестовая процедура восстановления в отдельный namespace.

Для консистентной файловой копии остановите запись в приложение или временно масштабируйте backend до нуля:

Пример: bash

```
kubectl -n pvt scale deployment/pvt-backend --replicas=0
# создать snapshot/backup средствами StorageClass
kubectl -n pvt scale deployment/pvt-backend --replicas=1
kubectl -n pvt rollout status deployment/pvt-backend --timeout=10m
```

После восстановления проверьте права доступа, наличие tree.json, EOS-кэш и акустические модели.

12. Диагностика

Frontend возвращает 404 при обновлении страницы

Nginx не настроен как SPA. Требуется `try_files $uri $uri/ /index.html`.

Запросы /api/* возвращают 404

Ingress не удаляет префикс /api. Проверьте regex path и rewrite-target: /\$2.

API возвращает 401

Проверьте совпадение PVT_API_KEY и значения, с которым был собран frontend. Не выводите ключ в лог.

Backend pod не стартует

Проверить:

- архитектуру узла;
- импорт pvteos, pr, srk в логе сборки;
- наличие /app/test.xlsx;
- права записи в /data/projects;
- OOMKilled и фактическое потребление памяти;

PVC не подключается при обновлении

Убедиться, что Deployment использует replicas: 1 и strategy.type: Recreate. Для RWO-тома новый pod не должен запускаться одновременно со старым.

Проекты теряются после перезапуска

Проверить PROJECTS_DIR=/data/projects, volumeMount и фактический путь StorageClass. Данные, записанные в /app/projects, являются эфемерными.

predict_ready **равен** false

Проверить /api/acoustic/status, наличие torch, meta.json и model.pt в каталоге _models на PVC.

13. Рекомендуемые доработки перед production

Приоритет P0:

- добавить `zod` в зависимости и добиться чистой frontend-сборки;
- добавить runtime-стадию frontend-образа;
- заменить временный пароль на серверную аутентификацию;
- добавить `/healthz` и `/readyz` без тяжёлых вычислений;
- зафиксировать одну реплику backend и настроить резервное копирование PVC;
- использовать immutable image tags/digests.

Приоритет P1:

- добавить метрики Prometheus: latency, 4xx/5xx, длительность EOS, размер очереди, свободное место PVC;
- добавить graceful shutdown и контролируемое завершение фоновых задач;
- определить retention для WAV, спектрограмм и model cache;
- выполнить нагрузочный тест и подобрать requests/limits.

Для горизонтального масштабирования backend потребуется внешнее транзакционное хранилище или распределённые блокировки. Простого подключения RWX PVC недостаточно.