

## # Разворачивание с билдом контейнеров

Инструкции, helm чарты и docker файлы для разворачивания сервисов.

Сервисы можно установить двумя способами.

- с помощью CI/CD Gitlab
- с помощью ручной установки Helm чартов

## # Зависимости

Для обоих способов установки:

- [K0s](https://k0sproject.io)
- [Helm](https://helm.sh)
- [Nginx Ingress](https://nginx.org)
- [ArgoCD](https://argo-cd.readthedocs.io/en/stable/getting\_started/)
- [Registry](https://help.sonatype.com/en/sonatype-nexus-repository.html)
- [python 3.12.11](https://www.python.org/downloads/release/python-31211/)

Для установки с помощью CI/CD Gitlab необходимо установить:

- [Gitlab](https://about.gitlab.com/install/)
- [Gitlab-Runner](https://docs.gitlab.com/runner/install/)

## ## Установка K0s, Helm, Nginx Ingress и ArgoCD

### ### Устанавливаем K0s

K0s - это облегченная копия K8s

```
```sh
curl -sSLf get.k0s.sh | sudo sh
```
```

Копируем `configs/k0s.yaml` в `/etc/k0s/k0s.yaml` и меняем:

- `spec.api.address`
- `spec.api.sans`
- `spec.storage.etcd.peer_address`

Создаем systemd сервис

```
```systemd
[Unit]
Description="k0s server"
After=network-online.target
Wants=network-online.target
```

```
[Service]
Type=simple
ExecStart=/usr/local/bin/k0s server -c /etc/k0s/k0s.yaml --enable-worker --kubelet-extra-args="--node-ip=LOCAL_IP_HERE"
Restart=always
```

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

```

Копируем конфиг для доступа к кубернетису и делаем alias для удобной работы

```
```sh
sudo cat /var/lib/k0s/pki/admin.conf > .kube/config.yaml
alias kubectl="k0s kubectl"
export KUBECONFIG=.kube/config.yaml
...

```

Делаем untaint ноде, т.к она у нас одна

```
```sh
kubectl taint nodes NODE_NAME node-role.kubernetes.io/master:NoSchedule-
...

```

### Устанавливаем Helm

Helm - это менеджер чартов для k0s

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

```

### Устанавливаем Nginx Ingress

```
```sh
kubectl label node NODE_NAME role=ingress
```

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm install mngress-nginx ingress-nginx/ingress-nginx \
  --set controller.kind=DaemonSet \
  --set controller.nodeSelector.role=ingress \
  --set controller.service.type=LoadBalancer
```

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
helm install cert-manager jetstack/cert-manager \
```

```
--set installCRDs=true  
...
```

Далее применяем ClusterIssuer для выпуска сертификатов

Создаем файл clusterIssuer.yaml

```
```yaml  
apiVersion: cert-manager.io/v1  
kind: ClusterIssuer  
metadata:  
  name: letsencrypt-prod  
spec:  
  acme:  
    server: https://acme-v02.api.letsencrypt.org/directory  
    email: EMAIL  
    privateKeySecretRef:  
      name: letsencrypt-prod  
    solvers:  
      - http01:  
        ingress:  
          class: nginx  
...`
```

```
```sh  
kubectl apply -f clusterIssuer.yaml  
...`
```

### Устанавливаем ArgoCD

```
```sh  
kubectl create namespace argocd  
  
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-  
cd/stable/manifests/install.yaml  
...`
```

И создаем ingress для ArgoCD

```
```yaml  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  labels:  
    app.kubernetes.io/name: argocd-cmd-params-cm  
    app.kubernetes.io/part-of: argocd  
  name: argocd-cmd-params-cm  
data:
```

```
server.insecure: "true"
```

```
---
```

```
apiVersion: networking.K0s.io/v1
kind: Ingress
metadata:
  name: argo-ingress
  namespace: argocd
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - argocd.DOMAIN
    secretName: argocd-tls
  rules:
  - host: argocd.DOMAIN
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: argocd-server
            port:
              number: 80
```

```
---
```

### ### Установка Registry

В качестве registry можно использовать Nexus или Harbor или другой доступный docker/pipx registry.

Инструкция по установке и конфигурировании

[Nexus](<https://help.sonatype.com/en/sonatype-nexus-repository.html>)

### ### Установка python

Требуется python версии 3.12.11

[python@3.12.11](<https://www.python.org/downloads/release/python-31211/>)

### ## Установка Gitlab и Gitlab-Runner

Установка выполняется согласно стандартным инструкциям:

[Install GitLab](<https://about.gitlab.com/install/>)

## # Установка инфраструктурных сервисов

Предварительные настройки:

- namespace
- GPU Operator

Инфраструктурными сервисами являются:

- OpenWebUI
- Milvus
- Kafka
- llamacpp

Установка инфраструктурных сервисов производится при помощи helm чартов и values файлов

### ## Установка namespace

```
``sh
kubectl create namespace infra
kubectl create namespace gpu-operator
``
```

### ## Установка GPU Operator

#### ## Установка GPU Operator (для любых GPU)

##### ### 1. Установить драйверы GPU на хостах

Перед установкой оператора необходимо, чтобы на каждом узле с GPU были установлены и корректно работали драйверы видеокарты (в зависимости от производителя и ОС).

- Для Astra Linux: используйте официальные инструкции по установке драйверов для вашей модели GPU
- Для Ubuntu: используйте официальные инструкции по установке драйверов для вашей модели GPU

Проверка: на узле должна быть доступна утилита/команда производителя или системные средства, подтверждающие, что устройство определяется ОС и драйвер загружен.

---

##### ### 2. Установить Container Toolkit для GPU

Установите пакет(ы), обеспечивающие проброс GPU в контейнеры (container runtime integration). Способ установки зависит от производителя GPU и репозитория вашей ОС.

---

### ### 3. Сконфигурировать containerd для работы с GPU

1) Добавьте конфигурацию runtime для GPU в containerd:

- Скопируйте файл `configs/gpu.toml` в каталог `containerd.d/gpu.toml` (путь и имя файла — согласно принятой структуре проекта).

2) Для Astra Linux может потребоваться дополнительная настройка runtime-конфигурации (в зависимости от реализации драйвера/toolkit).

Если в вашей среде требуется отключение управления sgroups со стороны GPU runtime, внесите параметр:

```
``toml
[gpu-container-cli]
no-cgroups = true
````
```

Примечание: точный путь к конфигурационному файлу runtime и имя секции зависят от используемого GPU toolkit.

---

### ### 4. Перезапустить containerd и кластерный сервис

Перезапустите:

- `containerd`
- сервис/процесс Kubernetes-дистрибутива (например, k0s), если он используется

---

### ### 5. Установить Helm Chart с GPU Operator

1) Добавьте репозиторий Helm и обновите индекс:

```
``sh
helm repo add gpu-operator-repo <URL_РЕПОЗИТОРИЯ_GPU_OPERATOR> && helm
repo update
````
```

2) Установите GPU Operator в отдельный namespace:

```
``sh
helm install --wait gpu-operator \
-n gpu-operator --create-namespace \
gpu-operator-repo/gpu-operator \
--set driver.enabled=false \
```

```
--set toolkit.enabled=false \  
--set devicePlugin.config.name=time-slicing-config-fine  
```\n
```

Пояснение параметров:

- `driver.enabled=false` — драйверы ставятся вручную на хостах
- `toolkit.enabled=false` — container toolkit ставится вручную на хостах
- `devicePlugin.config.name=...` — выбор конфигурации плагина устройств (если используется)

---

### 6. Включить time-slicing (если требуется)

1) Примените конфигурацию time-slicing:

```
```\nsh  
kubectl apply -f configs/timeslicing.yaml  
```\n
```

2) Назначьте конфигурацию плагина на нужный узел:

```
```\nsh  
kubectl label node {node_name} gpu.com/device-plugin.config=any  
```\n
```

Примечание: ключ лейбла (`gpu.com/device-plugin.config`) должен соответствовать используемому device plugin в вашем GPU Operator. Если в вашем чарте используется иной ключ, замените его на актуальный.

## Установка OpenWebUI

```
```\nsh  
helm upgrade --install open-webui open-webui/open-webui -f charts/open-webui/values-  
override.yaml -n infra  
```\n
```

Создаем файл ingressOpenWebUI.yaml

```
```\nyaml  
apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
  name: open-webui-ingress  
  namespace: infra  
  annotations:  
    cert-manager.io/cluster-issuer: "letsencrypt-prod"  
```\n
```

```
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - webui.DOMAIN
    secretName: webui-tls
  rules:
  - host: webui.DOMAIN
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: open-webui
            port:
              number: 8000
    ...

```sh
kubectl apply -f ingressOpenWebUI.yaml
```
```

```
## Milvus
```

```
...
helm upgrade --install milvus zilliztech/milvus -f charts/milvus/values-override.yaml -n infra
```
```

```
## Kafka
```

```
...
helm upgrade --install kafka oci://registry-1.docker.io/bitnamicharts/kafka -f
charts/kafka/values-override.yaml -n infra
```
```

```
## Llamacpp
```

1. Скачиваем модель

[Qwen3-32B]([https://huggingface.co/Qwen/Qwen3-32B-GGUF/blob/main/Qwen3-32B-Q4\\_K\\_M.gguf](https://huggingface.co/Qwen/Qwen3-32B-GGUF/blob/main/Qwen3-32B-Q4_K_M.gguf))

1. В файле /charts/llamacpp/values.yaml меняем hostpath и arg.model пути в соответствии с тем, куда загрузили модель.

При этом arg.model - путь должен начинаться с /models, т.к hostpath монтируется в /models

```
```sh
helm upgrade --install llamacpp-server charts/llamacpp -n infra
```
```

# Подготовка и сборка базовых образов

## Подготовка к сборке базовых образов

Необходимо собрать два базовых образа

- cuda128:3.12

- base:3.12

Докер файл лежат в base-images.

При сборке базового образа будут использоваться маршруты в сеть Интернет:

- docker.io
- registry-1.docker.io
- production.cloudflare.docker.com
- security.ubuntu.com
- archive.ubuntu.com
- keyserver.ubuntu.com
- ppa.launchpadcontent.net
- bootstrap.pyupio.com
- pypi.org
- download.pytorch.org

## Сборка базовых образов

```
```sh
```

```
# gpu-runtime:py3.12
```

```
docker build -t $CI_REGISTRY/gpu-runtime:py3.12 \
--build-arg BASE_IMAGE=<GPU_RUNTIME_BASE_IMAGE> \
--build-arg PYTHON_VER="3.12" .
```

Где:

- `<GPU_RUNTIME_BASE_IMAGE>` — базовый runtime-образ с нужными библиотеками для работы GPU в контейнере (выбирается под вашу платформу/производителя).
- `gpu-runtime:py3.12` — универсальный тег вместо `cuda128:3.12`.

```
# base:3.12
```

```
docker build -t $CI_REGISTRY/base:3.12 --build-arg BASE_IMAGE=ubuntu:noble --build-arg PYTHON_VER="3.12" .  
```
```

И выгружаем в свой registry командами:

```
```sh  
docker push $CI_REGISTRY/cuda128:3.12  
docker push $CI_REGISTRY/base:3.12  
```
```

### # Подготовка и сборка сервисов

Используя полученные базовые образы, необходимо собрать сервисы которые собираются так же через Docker без указания переменных

### ## Подготовка к сборке сервисов

При сборке сервисов использоваться маршруты в сеть Интернет:

- docker.io
- registry-1.docker.io
- production.cloudflare.docker.com
- security.ubuntu.com
- archive.ubuntu.com
- keyserver.ubuntu.com
- ppa.launchpadcontent.net
- bootstrap.pyup.io
- pypi.org
- download.pytorch.org

Необходимые python пакеты, которые необходимо положить в свой артефакторий для сборки без интернета

Находятся в файле ```requirements.lock```

Сборку можно осуществлять двумя способами:

- вручную
- с помощью CI/CD Gitlab

### ## Сборка сервисов вручную

Для сборки вручную нам понадобятся

- ```CI\_REGISTRY\_USERNAME``` - логин registry
- ```CI\_REGISTRY\_PASSWORD``` - пароль registry

- ``CI\_REGISTRY`` - URL к registry
- ``CI\_PYPI\_HOSTED`` - URL к pypi registry
- ``CI\_PYPI\_REGISTRY`` - URL к pypi registry

1. Клонирование репозиторий сервисов к себе, а именно:

- Библиотеки:

- ``git pull https://gitverse.ru/evamultiagent/plib`` - Набор часто используемых функций.
- ``git pull https://gitverse.ru/evamultiagent/chaincore`` - Фреймворк для оркестрации
- ``git pull https://gitverse.ru/evamultiagent/responder`` - Фреймворк для работы с LLM через API llama.cpp или OpenAI протокол
- ``git pull https://gitverse.ru/evamultiagent/vectorcore`` - Фреймворк для организации хранения и поиска для RAG
- ``git pull https://gitverse.ru/evamultiagent/recognitor`` - Фреймворк LR (layout recognition), TR (table recognition), OCR (Optical Character Recognition) с упором на качественное распознавание таблиц

- Клонирование репозиторий Core сервисов:

- ``git pull https://gitverse.ru/evamultiagent/nlpilot`` - Сервис оркестрации - пример для NLP+LLM пайплайнов
- ``git pull https://gitverse.ru/evamultiagent/hippocampus`` - Сервис запоминания и поиска знаний.
- ``git pull https://gitverse.ru/evamultiagent/docfiller`` - Сервис работы с документами
- ``git pull https://gitverse.ru/evamultiagent/filevault`` - Сервис доступа к файлам
- ``git pull https://gitverse.ru/evamultiagent/medulla`` - Сервис который работает с LLM через API llama.cpp или OpenAI протокол
- ``git pull https://gitverse.ru/evamultiagent/paleograph`` - Сервис парсинга документов

- Клонирование (``git pull``) репозиторий сервисов интеграций:

- ``git pull https://gitverse.ru/evamultiagent/conflu-gate`` - Сервис с интеграции с Конфлюенс
- ``git pull https://gitverse.ru/evamultiagent/xchange-gate`` - Сервис интеграции с Ms Exchange

2. Создаем секреты в K0s

- hippocampus

```
`` kubectl create secret generic hippocampus-secret --from-literal=openai_key="openai_key" --from-literal=openai_url="openai_url" --from-literal=llamacpp_url="llamacpp_url" -n evamultiagent ``
```

- medulla

```
`` kubectl create secret generic hippocampus-secret --from-literal=openai_key="openai_key" --from-literal=openai_url="openai_url" --from-literal=llamacpp_url="llamacpp_url" -n evamultiagent ``
```

- conflu-gate

```
``` kubectl create secret generic conflu-gate-secret --from-literal=cfl_token="token" --from-literal=cfl_url="url" -n evamultiagent ```
```

- xchange-gate

```
``` kubectl create secret generic xchange-gate-secret --from-literal=email="user@example.com" --from-literal=host="smtp.example.com" --from-literal=login="mylogin" --from-literal=password="mypassword" -n evamultiagent ```
```

### 3. Собираем SDK пакеты в сервисах

Устанавливаем пакеты сборки для python версии 3.12.11

```
```sh  
pip install build twine  
```
```

Заходим в каждую библиотеку и собираем её:

```
```sh  
python -m build  
python -m twine upload --repository-url ${CI_PYPI_HOSTED} -u ${CI_REGISTRY_USER} -p  
${CI_REGISTRY_PASSWORD} dist/*  
```
```

Заходим в каждый Core сервис и в каждый сервис интеграций и собираем:

```
```sh  
cd ./sdk  
python -m build  
python -m twine upload --repository-url ${CI_PYPI_HOSTED} -u ${CI_REGISTRY_USER} -p  
${CI_REGISTRY_PASSWORD} dist/*  
```
```

И собираем Docker образы

```
```sh  
docker build -t ${CI_REGISTRY}/evamultiagent/REPO_NAME:main \  
--build-arg pypi_index_url=${CI_PYPI_REGISTRY} .  
docker push ${CI_REGISTRY}/evamultiagent/REPO_NAME:main  
```
```

### ## Сборка при помощи Gitlab

1. В ``GitLab CI\CD ENV`` добавляем

- ``CI\_REGISTRY\_USERNAME`` - логин registry
- ``CI\_REGISTRY\_PASSWORD`` - пароль registry
- ``CI\_REGISTRY`` - URL к registry

- ``CI\_PYPI\_HOSTED`` - URL к pypi registry
- ``CI\_PYPI\_REGISTRY`` - URL к pypi registry
- ``CI\_ARGOCLI\_IMAGE`` - ``dh-mirror.gitverse.ru/zebrains/argocd-cli:main``
- ``GITVERSE\_UPLOAD\_ALLOWED`` = false

2. Клонировать репозитории сервисов к себе в GitLab, а именно:

- Библиотеки:
  - [plib](https://gitverse.ru/evamultiagent/plib) - Набор часто используемых функций.
  - [chaincore](https://gitverse.ru/evamultiagent/chaincore) - Фреймворк для оркестрации
  - [responder](https://gitverse.ru/evamultiagent/responder) - Фреймворк для работы с LLM через API llama.cpp или OpenAI протокол
  - [vectorcore](https://gitverse.ru/evamultiagent/vectorcore) - Фреймворк для организации хранения и поиска для RAG
  - [recognitor](https://gitverse.ru/evamultiagent/recognitor) - Фреймворк LR (layout recognition), TR (table recognition), OCR (Optical Character Recognition) с упором на качественное распознавание таблиц

- Клонировать репозитории Core сервисов:

- [nlpilot](https://gitverse.ru/evamultiagent/nlpilot) - Сервис оркестрации - пример для NLP+LLM пайплайнов
- [hippocampus](https://gitverse.ru/evamultiagent/hippocampus) - Сервис запоминания и поиска знаний.
- [docfiller](https://gitverse.ru/evamultiagent/docfiller) - Сервис работы с документами
- [filevault](https://gitverse.ru/evamultiagent/filevault) - Сервис доступа к файлам
- [medulla](https://gitverse.ru/evamultiagent/medulla) - Сервис который работает с LLM через API llama.cpp или OpenAI протокол
- [paleograph](https://gitverse.ru/evamultiagent/paleograph) - Сервис парсинга документов

- Клонировать репозитории сервисов интеграций:

- [conflu-gate](https://gitverse.ru/evamultiagent/conflu-gate) - Сервис с интеграции с Конфлюенс
- [xchange-gate](https://gitverse.ru/evamultiagent/xchange-gate) - Сервис интеграции с Ms Exchange

3. Выставляем в GitLab CI/CD Settings каждого сервиса Variables:

**\*\*conflu-gate:\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/base:3.12

**\*\*xchange-gate\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/base:3.12

**\*\*docfiller\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/base:3.12

**\*\*hippocampus\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/cuda128:3.12

**\*\*medulla\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/base:3.12

**\*\*nlpilot\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/base:3.12

**\*\*paleograph\*\***

- BV\_BASE\_IMAGE: \$CI\_REGISTRY/cuda128:3.12

#### 4. Создаём теги для каждого сервиса

```
```sh
git tag v1.0.0
git push origin v1.0.0
```
```

Важно! После тегирования и пуша каждого репозитория начнётся сборка. В процессе сборки можно перейти к дальнейшим пунктам.

#### 5. В K0s создаем секреты

**\*\*conflu-gate:\*\***

```
``` kubectl create secret generic conflu-gate-secret --from-literal=cfl_token="token" --from-literal=cfl_url="url" -n evamultiagent ```
```

**\*\*xchange-gate\*\***

```
``` kubectl create secret generic xchange-gate-secret --from-literal=email="user@example.com" --from-literal=host="smtp.example.com" --from-literal=login="mylogin" --from-literal=password="mypassword" -n evamultiagent ```
```

**\*\*hippocampus\*\***

```
``` kubectl create secret generic hippocampus-secret --from-literal=openai_key="openai_key" --from-literal=openai_url="openai_url" --from-literal=llamacpp_url="llamacpp_url" -n evamultiagent ```
```

**\*\*medulla\*\***

```
``` kubectl create secret generic hippocampus-secret --from-literal=openai_key="openai_key" --from-literal=openai_url="openai_url" --from-literal=llamacpp_url="llamacpp_url" -n evamultiagent ```
```

6. Ожидаем сборки каждого сервиса

# Запуск сервисов

## Настройка launchpad через Gitlab

1. Клонировем - [launchpad](https://gitverse.ru/evamultiagent/launchpad) к себе в gitlab

2. в ``GitLab CI\CD ENV`` добавляем

- ``ARGOCD\_URL`` - url ArgoCD
- ``ARGOCD\_USERNAME`` - логин ArgoCD
- ``ARGOCD\_PASSWORD`` - пароль ArgoCD

3. в ``configs/main.yaml`` в соответствии со своим сервером поменять поиском с заменой

``registry.url: https://gitverse.ru/evamultiagent/SERVICE\_NAME.git`` на  
``https://GITLAB\_URL/evamultiagent/SERVICE\_NAME.git``

``image.url: dh-mirror.gitverse.ru/zebrains/SERVICE\_NAME`` на ``image.url:  
CI\_REGISTRY/evamultiagent/SERVICE\_NAME``

``pathsPassthrough.nodeName: "astra"`` на ``pathsPassthrough.nodeName:  
"SERVERNAME"``

``pathsPassthrough.host: "/mnt/big\_data/books"`` на ``pathsPassthrough.host: "Локация  
на сервере"``

4. Так же у filevault в ``configs/main.yaml`` выставим домен в kubernetes.ingress такой же, как и у open-webui

5. Коммитим и ожидаем разворачивания в ArgoCD

```
``sh
git commit -m 'values update'
git push origin v1.0.0
``
```

## Разворачивание вручную

1. Переменные, которые нам нужны:

- ``ARGOCD\_URL`` - url ArgoCD
- ``ARGOCD\_USERNAME`` - логин ArgoCD
- ``ARGOCD\_PASSWORD`` - пароль ArgoCD
- ``CI\_REGISTRY\_USERNAME`` - логин registry

- ``CI\_REGISTRY\_PASSWORD`` - пароль registry
- ``CI\_REGISTRY`` - URL к registry

## 2. Запускаем сервисы:

Меняем в charts/local переменные:

- REPO\_URL, PATH\_ON\_HOST во всех сервисах
- INGRESS\_URL в сервисах nlpilot и filevault и запускаем сервис.

где:

- REPO\_URL - ``url к вашему git``
- PATH\_ON\_HOST - ``путь к директории на сервере``
- INGRESS\_URL - ``внешний url``

**\*\*nlpilot\*\***: Изменить REPO\_URL и PATH\_ON\_HOST, INGRESS\_URL

```
``sh
helm install nlpilot ./nlpilot -f charts/local/nlpilot.values-override.yaml -n evamultiagent
``
```

**\*\*conflu-gate\*\***: Изменить REPO\_URL и PATH\_ON\_HOST

```
``sh
helm install conflu-gate ./conflu-gate -f charts/local/conflu-gate.values-override.yaml -n
evamultiagent
``
```

**\*\*docfiller\*\***: Изменить REPO\_URL и PATH\_ON\_HOST

```
``sh
helm install docfiller ./docfiller -f charts/local/docfiller.values-override.yaml -n evamultiagent
``
```

**\*\*filevault\*\***: Изменить REPO\_URL и PATH\_ON\_HOST, INGRESS\_URL

```
``sh
helm install filevault ./filevault -f charts/local/filevault.values-override.yaml -n evamultiagent
``
```

**\*\*hippocampus\*\***: Изменить REPO\_URL и PATH\_ON\_HOST

```
``sh
helm install hippocampus ./hippocampus -f charts/local/hippocampus.values-override.yaml -
n evamultiagent
``
```

**\*\*medulla\*\***: Изменить REPO\_URL и PATH\_ON\_HOST

```
```sh
```

```
helm install medulla ./medulla -f charts/local/medulla.values-override.yaml -n evamultiagent
```

```
```
```

**\*\*paleograph\*\***: Изменить REPO\_URL и PATH\_ON\_HOST

```
```sh
```

```
helm install paleograph ./paleograph -f charts/local/paleograph.values-override.yaml -n evamultiagent
```

```
```
```

**\*\*xchange-gate\*\***: Изменить REPO\_URL и PATH\_ON\_HOST

```
```sh
```

```
helm install xchange-gate ./xchange-gate -f charts/local/xchange-gate.values-override.yaml -n evamultiagent
```

```
```
```