Общее описание

Описание продукта

ZeBrains AI Multi-agent — Корпоративная платформа оркестрации мультиагентных ИИ-систем для автоматизации бизнес-процессов предприятий. Обеспечивает развертывание специализированных агентов с минимальными затратами на разработку, полным соответствием стандартам информационной безопасности, поддержкой изолированной инфраструктуры (On-Premise) и интеграцией с локальными языковыми моделями большого объема (LLM). Ключевые технические преимущества:

Готовая архитектура для быстрого создания и развертывания AI-агентов;
Соответствие требованиям ИБ — полная изоляция данных и процессов;
Автономное развертывание — работа без зависимости от внешних сервисов;
Совместимость с локальными LLM — независимость от облачных решений.
Решение предназначено для организаций, требующих высокого уровня контроля над ИИ-инфраструктурой и обработкой корпоративных данных.

Сервис представляет собой самостоятельный компонент корпоративной ИТ-инфраструктуры, обеспечивающий реализацию повседневных бизнес-задач проектного офиса, автоматизацию базовых процессов работы с документами и и поиск информации по базе знаний в двух источниках - Confluence и сетевые папки Компании.

Применение в корпоративной среде

Мульти-агентные системы ZeBrains AI находят широкое применение в:

- Автоматизации HR-процессов;
- Управлении корпоративными данными;
- Координации рабочих процессов;
- Принятии стратегических решений;
- Автоматизация бизнес-процессов;
- Умные города и IoT;
- Анализ рисков, обработка транзакций и персонализированные финансовые рекомендации;
- Распределенные системы поддержки клиентов с различными специализированными агентами.

Развитие технологий мульти-агентных систем демонстрирует движение к более сложным архитектурам, где ZeBrains AI может интегрировать различные подходы к оркестрации агентов:

- Последовательная оркестрация для пошаговых процессов;
- Параллельная обработка для одновременного выполнения задач;
- Групповые обсуждения для коллективного решения проблем;
- Передача управления между специализированными агентами.

Важно отметить, что конкретная расшифровка ZeBrains AI может варьироваться в зависимости от реализации и целевого назначения конкретной мульти-агентной системы. В большинстве случаев выбор расшифровки определяется специфическими требованиями проекта и областью применения.

Функциональные требования

Получение данных по загруженной оргструктуре компании и базе знаний по документам

Система позволяет искать данные сотрудников по загруженному файлу оргструктуры Компании (при его наличии в источниках данных). При несоответствии входящих документов по формату и/или наполнению действия Системы не регламентируются.

Ввод запроса к ZeBrains AI и вывод ответа

Пользователь обращается к ZeBrains AI через web-интерфейс, вводя поисковой запрос. После ввода сообщения ZeBrains AI осуществляет поиск по проиндексированным источникам данных и отображает ответ на заданный вопрос. На каждый ответ Системы ZeBrains AI прикрепляет до 10 документов, которые считает семантически близкими источниками формирования ответа.

Общение Пользователя с интерфейсом ZeBrains AI происходит при помощи естественного языка. Формат отправки запроса на выдачу результата - свободный.

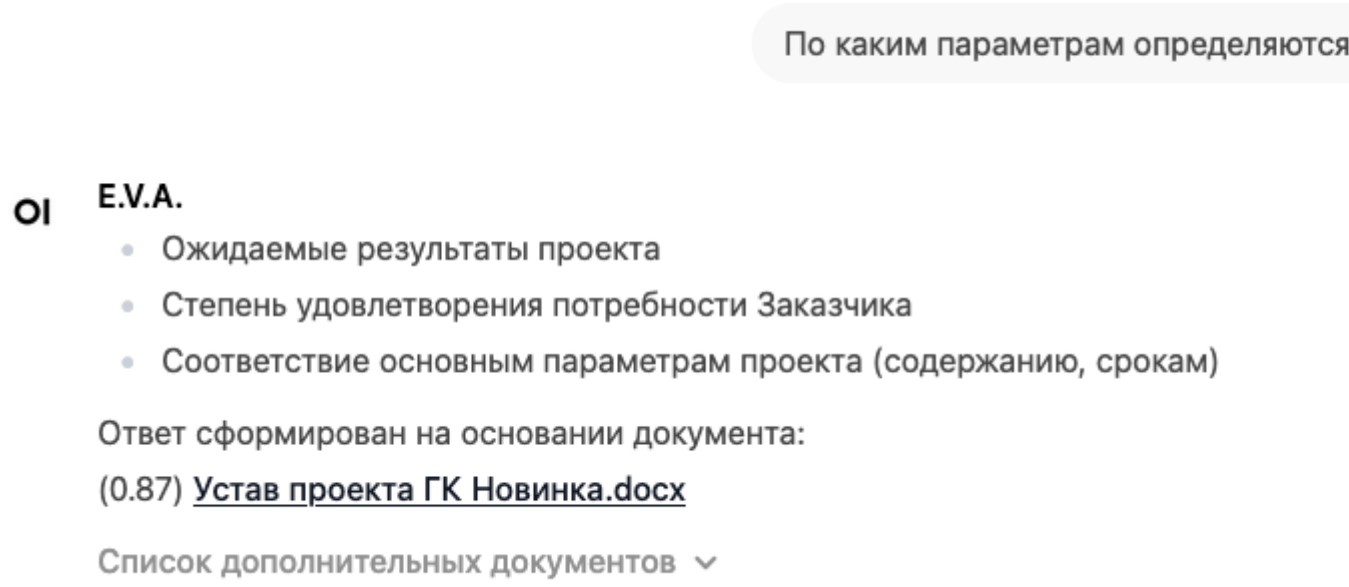

Рис 1. Вывод сообщения по заданному вопросу

ZeBrains AI позволяет скачивать любой из файлов, приложенных к ответу. Скачивание на устройство Пользователя осуществляется автоматически при клике на название приложенного документа

Получение дополнительной информации от ZeBrains AI

Если ZeBrains AI предлагает краткий ответ, либо если Пользователю необходим более подробный/развёрнутый ответ, допустимо использование вспомогательных команд (например, “Расскажи подробнее”, “Переформулируй ответ”, “Выведи дополнительную информацию” и др”). Допустимо также уточнение контекста поиска для Системы путём указания источника /или документа, по которому ZeBrains AI должна предложить ответ.

Сохранение контекста диалога

Система держит в памяти контекст текущего диалога с пользователем. Для более корректных ответов и/или уточнения выводимых данных рекомендуется сбрасывать контекст общения с ZeBrains AI Сброс контекста осуществляется командой “Забудь”. При вызове команды “Забудь” ZeBrains AI выводит в ответном сообщение уведомление о том, что предыдущий контекст сброшен и Система готова начать новый диалог с Пользователем.

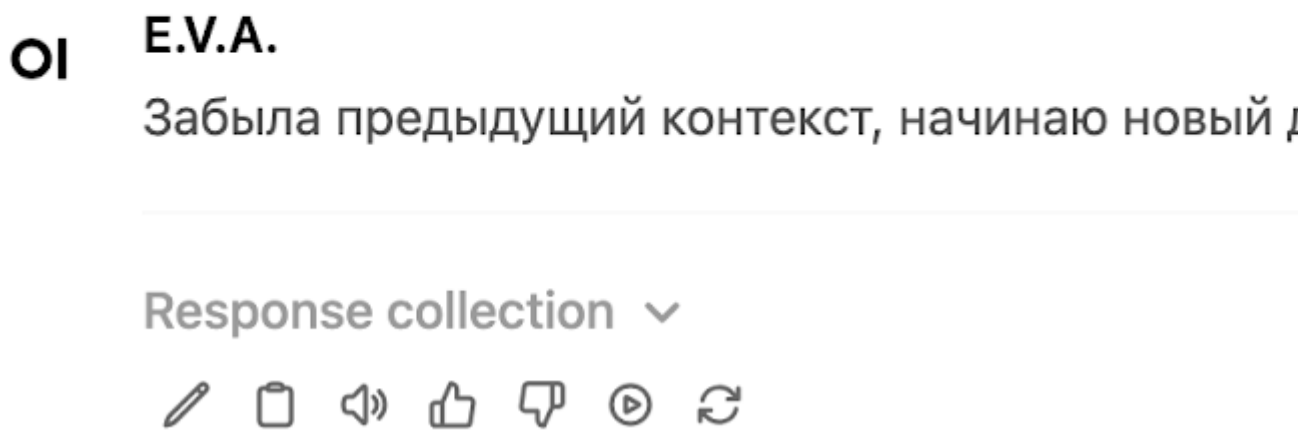

Рис 2. Пример сброса контекста диалога

Рекомендуется разграничивать контекст поисковых запросов к ZeBrains AI в разных чатах (например, не запрашивать информацию по различным областям знаний в рамках одной диалоговой сессии), однако это требование не строго регламентировано и может быть обработано путём вызова команды “Забудь” для сброса предыдущего контекста.

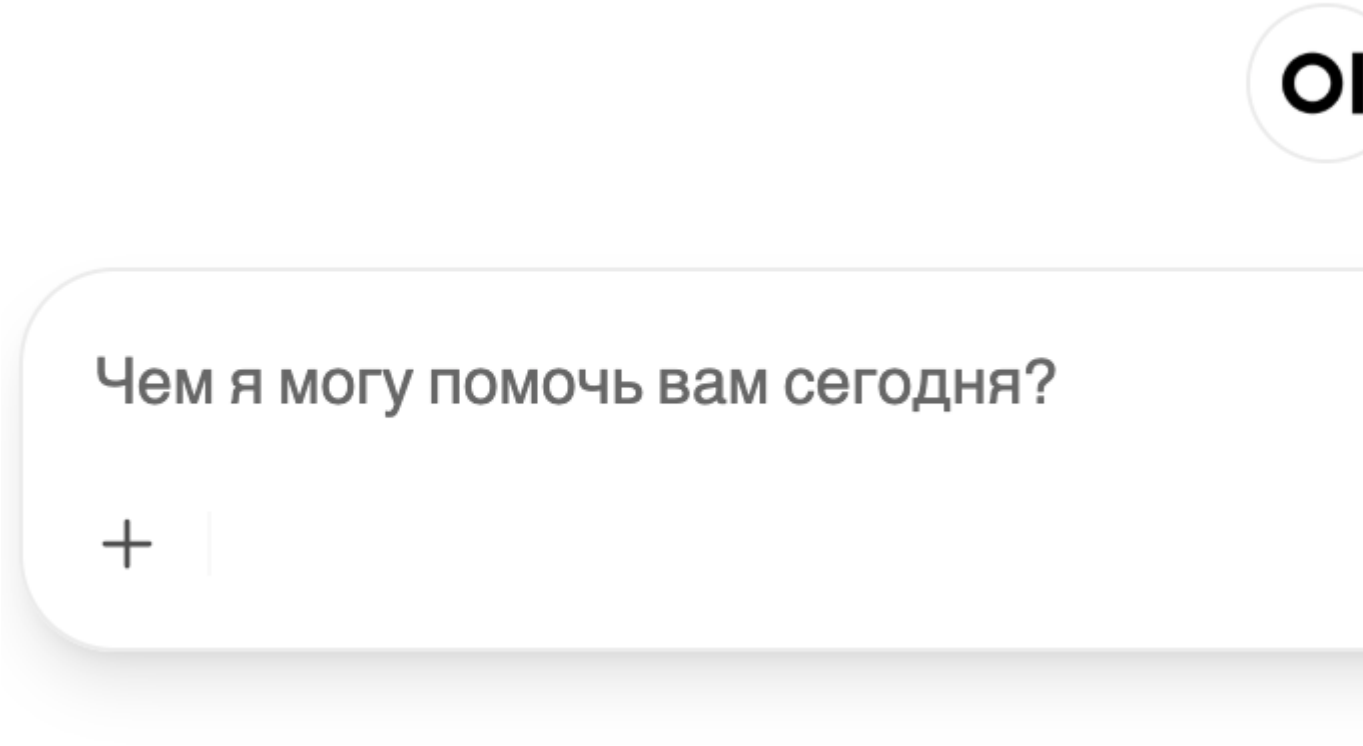

Рис 3. Пример начала нового чата с ZeBrains AI

Неточности или неполнота ответов

ZeBrains AI допускает в работе ошибки, и это нормальный регламентированный процесс. Если запрос не отработал согласно ожиданиям пользователя, ZeBrains AI можно скорректировать, сузив область поиска знаний свободным промптом к Системе.

В случае, если от ZeBrains AI получен неполный или некорректный ответ, рекомендуется скорректировать выдачу с помощью дополнительного уточняющего запроса либо сброса контекста диалога (см. выше). Если после этого ZeBrains AI всё равно даёт неполный/некорректный ответ, рекомендуется обратиться в ИТ отдел и описать сценарий, с которым ZeBrains AI не справилась.

Дополнительные функции ZeBrains AI

ZeBrains AI позволяет инициировать ряд дополнительных функций при работе с данными. Контекст и применимость зависят от формата и типа данных, которые Пользователь запрашивает у ZeBrains AI Например, ZeBrains AI может построить Gantt-диаграмму активностей, сформировать memraid-диаграмму рисков, выделить шаблон документа из уже заполненного документа, построить план реализации проекта на основании входных данных в тексте. Запрос с дополнительными функциями и возможностями к ZeBrains AI осуществляется в свободной форме.

Требования к данным

Тип данных

ZeBrains AI может работать с различными типами данных, как со структурированными, так и неструктурированными. В качестве структурированных входных данных выступают файлы в формате XLSX, XLS. В качестве неструктурированных – файлы счетов, договоров (DOC, DOCX, TXT), презентации (PPTX) и сканы документов в формате PDF. Файлы изображений (JPG, PNG, TIFF и др) не допускаются.

Качество данных

Документы, из которых предполагается извлечение данных, должны быть установленного качества и определенной структуры. Под установленным качеством документов понимается следующее:

1. Текст должен быть напечатан стандартными шрифтами (не рукописный, не стилизованный декоративный).
2. Отсутствие размытий, смазываний или нерезкости текста.
3. Отсутствие перекрытий текста другими элементами (рисунки, штампы, рукописные пометки, водяные знаки).
4. Отсутствие наложений, разреженности, физических повреждений документа, закрывающих текст (сколы, разрывы, складки, пятна, отсутствие части текста, наличие пометок, рисунков и пр. поверх распознаваемого текста, низкое качество текста).
5. Документы не должны иметь цифрового шума, цветовых искажений (RGB модель в приоритете), изменения геометрии методами трансформации и искажения.

Под определенной структурой документов, представленных в виде таблиц, подразумевается наличие области значений и заголовков (при необходимости, многоуровневых) с достоверным описанием данных, которые представлены в полях таблицы. При расположении таблицы на нескольких страницах документа, на каждой странице должен быть отображён заголовок данной таблицы.

Если документы не отвечают вышеописанным требованиям к качеству и структуре, действия системы не регламентируются.

Инструкция по разворачиванию ZeBrains AI

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

Установка K0s

K0s - это облегченная копия K8s

```
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

```
[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

```
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

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

Установка Helm

Helm - это менеджер чартов для k0s

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

Установка Nginx Ingress

```
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

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

Установка ArgoCD

```
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

```
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/pipy registry.
Инструкция по установке и конфигурировании Nexus

Установка python

Требуется python версии 3.12.11 python@3.12.11

Установка Gitlab и Gitlab-Runner

Установка выполняется согласно стандартным инструкциям: Install GitLab

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

```
kubectl create namespace infra

kubectl create namespace gpu-operator
```

Установка GPU Operator

1. Необходимо установить драйвера nvidia

Для Astra Linux установка выглядит следующим образом: Драйверы видеокарт Nvidia для Astra Linux на платформе x86-64
Для Ubuntu следующим: Установка драйверов NVIDIA и CUDA на Ubuntu

2. Далее ставим Nvidia Container toolkit

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o

/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg && curl -s -L

https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | sed

's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]

https://#g' | sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo apt update

sudo apt install -y nvidia-container-toolkit
```

3. Конфигурируем containerd
   - Копируем configs/nvidia.toml в containerd.d/nvidia.toml
   - Если используется AstraLinux, необходимо изменить параметры в /etc/nvidia-container-runtime/config.toml

```
[
nvidia-container-cli
]
no-cgroups = true
```

4. Перезапускаем containerd и K0s
5. Устанавливаем Helm Chart с gpu-opertator

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia && helm repo update
helm install --wait gpu-operator \
 -n gpu-operator --create-namespace \
  nvidia/gpu-operator \
  --set driver.enabled=false \
  --set toolkit.enabled=false \
  --set devicePlugin.config.name=time-slicing-config-fine
```

6. Добавляем time slicing

```
kubectl apply -f configs/timeslicing.yaml
kubectl label node {node_name} nvidia.com/device-plugin.config=any
```

Установка OpenWebUI

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

Создаем файл ingressOpenWebUI.yaml

```
apiVersion: networking.K0s.io/v1
kind: Ingress
metadata:
  name: open-webui-ingress
  namespace: infra
  annotations:
```

```
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
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
```

```
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

1. В файле /charts/llamacpp/values.yaml меняем hostpath и arg.model пути в соотвествии с тем, куда загрузили модель. При этом arg.model - путь должен начинаться с /models, т.к hostpath монтируется в /models

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

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
- developer.download.nvidia.com
- keyserver.ubuntu.com
- ppa.launchpadcontent.net
- bootstrap.pypa.io
- pypi.org
- download.pytorch.org

Необходимые python пакеты, которые необходимо положить в свой артифакторий 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. Клонируем репозитории 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 распознование таблиц

- Клонируем репозитории 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
- Клонируем (`git pull`) репозитории 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 сбрки для python версии 3.12.11
pip install build twine

Заходим в каждую библиотеку и собираем её:

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:

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

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. Клонируем репозитории сервисов к себе в GitLab, а именно:
- Библиотеки:
  - plib - Набор часто используемых функций.
  - chaincore - Фреймворк для оркестрации
  - responder - Фреймворк для работы с LLM через API llama.cpp или OpenAI протокол
  - vectorcore - Фреймворк для организации хранения и поиска для RAG
  - recognitor - Фреймворк LR (layout recognition), TR (table recognition), OCR (Optical Character Recognition) с упором на качественное распознование таблиц
- Клонируем репозитории Core сервисов:
  - nlpilot - Сервис оркестрации - пример для NLP+LLM пайплайнов
  - hippocampus - Сервис запоминания и поиска знаний.
  - docfiller - Сервис работы с документами
  - filevault - Сервис доступа к файлам
  - medulla - Сервис который работает с LLM через API llama.cpp или OpenAI протокол
  - paleograph - Сервис парсинга документов
- Клонируем репозитории сервисов интеграций: - conflu-gate - Сервис с интеграции с Конфлюенс - 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

`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. Клонируем - 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 серере"`

4. Так же у filevault в `configs/main.yaml` выставлем домен в kubernetes.ingress такой же, как и у open-webui
5. Коммитим и ожидаем разворачивания в ArgoCD

`git commit -m 'values update'`

`git push origin v1.0.0`

Развертывание 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

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

conflu-gate: Изменить REPO_URL и PATH_ON_HOST

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

docfiller: Изменить REPO_URL и PATH_ON_HOST

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

filevault: Изменить REPO_URL и PATH_ON_HOST, INGRESS_URL

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

hippocampus: Изменить REPO_URL и PATH_ON_HOST

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

medulla: Изменить REPO_URL и PATH_ON_HOST

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

paleograph: Изменить REPO_URL и PATH_ON_HOST

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

xchange-gate: Изменить REPO_URL и PATH_ON_HOST

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