Годовикова 9с17
7 495 234 2222
info@skala-

Описание архитектуры программного обеспечения для электронно-вычислительных машин «Скала^р Спектр ИИ»

Москва 2026 г.

скала^р

Годовикова 9с17
7 495 234 2222
info@skala-

Описание архитектурного стиля

Система имеет модульную архитектуру, обеспечивающую слабую связность компонентов.

Каждый функциональный блок (модуль) реализован как независимый сервис или компонент с четко определенными границами контекста.

Компоненты системы отчуждаемыми. Это подразумевает, что модуль может быть выделен в отдельный проект или передан другой команде/системе без потери функциональности и без необходимости модификации ядра системы.

Обеспечена возможность замены одного модуля на аналогичный (например, смена драйвера хранилища, провайдера вычислений или движка виртуализации) через использование контрактов и абстрактных интерфейсов. Реализация модуля не влияет на работу остальной части системы при соблюдении контракта.

Контейнеризация и изоляция

Для обеспечения изолированной среды исполнения, воспроизводимости сборки и удобства развертывания, система и ее компоненты используют контейнеризацию.

В качестве базовой технологии контейнеризации используется Docker.

Каждый компонент (или группа связанных компонентов) функционирует в рамках изолированных контейнеров. Это необходимо для обеспечения независимости окружения, ограничения потребляемых ресурсов (CPU, RAM, IO) и повышения безопасности.

Предусмотрена система оркестрации или средство управления жизненным циклом контейнеров (запуск, остановка, мониторинг состояния) в соответствии с бизнес-логикой приложения.

скала^р
Годовикова 9с17
7 495 234 2222
info@skala-

Асинхронность выполнения длительных операций

Операции, требующие значительного времени выполнения (длительные операции), не блокируют работу пользовательского интерфейса или основного потока управления приложения.

К длительным операциям относятся, но не ограничиваются: скачивание образов, запуск/остановка контейнеров, резервное копирование, импорт больших данных.

Для обеспечения асинхронного режима в системе реализована очередь задач (брокер сообщений, например, RabbitMQ, Kafka или Redis Queue).

При поступлении запроса на длительную операцию, система создаёт задачу, помещает её в очередь и немедленно возвращает клиенту подтверждение о принятии в работу (статус "принято").

Фоновые обработчики (воркеры) асинхронно забирают задачи из очереди, выполняют их, обновляют статус выполнения и уведомляют целевую систему (или клиента) о результате (успех/ошибка).

Верхнеуровневая архитектура

Архитектура предполагает разделение на отдельные независимые приложения и репозитарии (см. Рис.1), которые будут разделять общие системы хранения (БД Postgres, Redis):

- frontend,
- backend: backend + Celery (Celery workers) + Redis + Postgres,
- spectr_proxy,
- agent
- observability + LLM gateway,
- monitoring

Годовикова 9с17
7 495 234 2222
info@skala-

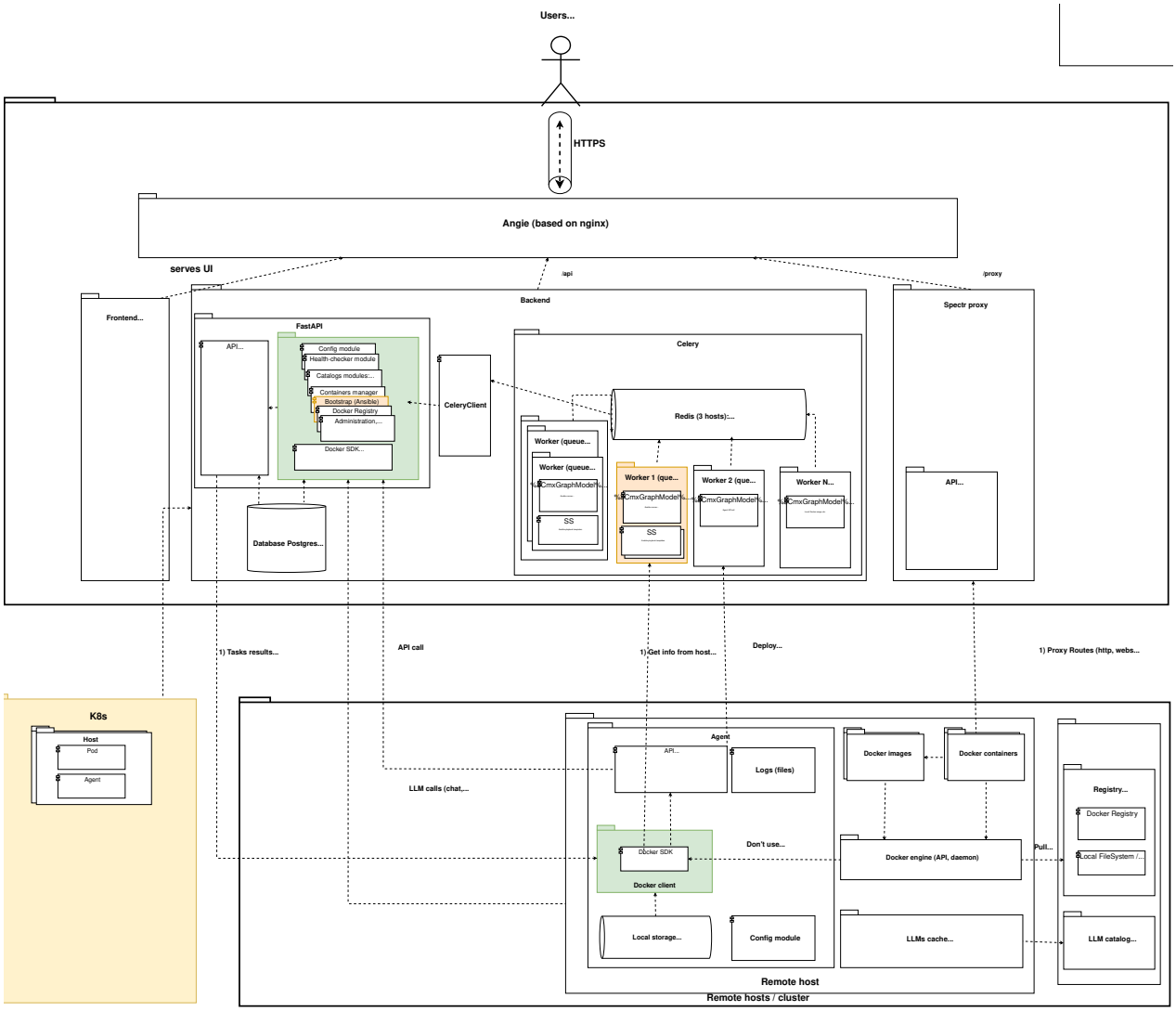

Рис.1 – Верхнеуровневая архитектура

Backend реализует ряд сервисов (модулей):

- администрирование и конфигурация Системы:
  - модуль администрирования,
  - модуль конфигурирования
- управление хостами:
  - Bootstrap (Ansible) в части подключения хостов и выполнение первоначального развертывания необходимых пакетов и компонентов;
  - мониторинг состояния хостов и контейнеров (health-check module);
- управление каталогами:
  - - управление каталогом LLM-моделей (Catalog: LLM);

Годовикова 9с17
7 495 234 2222
info@skala-

    - управление каталогом имиджей Docker (Catalog: docker images);
- управление приложениями - контейнерами Docker или многоконтейнерными приложениями Docker compose (containers manager);
- управление Docker Registry;
- управление проксированием (proxy module);
- управление Docker имиджами и контейнерами на хостах развертывания Спектр (Local Docker SDK module).

Агент обеспечивает выполнение следующих задач на хосте:

- исполнение команд по управлению имиджами и контейнерами: (docker images, docker pull/run/exec/stop/rm);
- получение логов работы с имиджами и выполнения контейнеров (`docker pull/run/exec`).

Компонент Celery используется:

- для снижения нагрузки со Спектр backend, передавая часть этой нагрузки на Celery workers. Это обеспечивает асихронное выполнение множества задач Спектр backend, освобождая его для выполнения функций управления Спектром;
- для гарантированного исполнения задач, в особенности задач с длительным временем исполнения, такими как deployment с Ansible;
- позволяет задавать политики retry в случае недоступности компонентов, с которыми взаимодействуют Celery workers.

Компонент Worker deploy (bootstrap) позволяет:

- использовать библиотеку ansible_runner для запуска Ansible playbooks,
- сохранять информацию о хосте, предоставляемую Ansible в БД,
- обеспечивать предоставление статуса (лога) развертывания Ansible playbooks.

Компонент Worker api обеспечивает взаимодействие с API Агента для запуска задач, связанных с Docker.

Компонент Worker'ы обеспечивают возможность retry в случае ошибок/недоступности Агента/недоступности сети.