Разработка облачно-ориентированного языкового сервера для Go для платформы SourceCraft 

Студент: Новокшанов Евгений
Руководитель: Ольга Лукьянова, Александр Захаров

Университет ИТМО
2025

IDE переезжает в облако

Тренд индустрии:

Разработка активно мигрирует из локальных машин в облачные среды (GitHub Codespaces, Gitpod). Платформа SourceCraft от Яндекса следует этому мировому тренду, интегрируя IDE напрямую в веб-интерфейс.

Новая парадигма:

В облачной IDE пользователь ожидает мгновенной работы "умных" функций (переход к определению, поиск использований) на любой версии кода (main, feature-ветка, старый коммит).

Актуальность:

Стандартные языковые серверы, такие как gopls(от Google), не были спроектированы для этого сценария. Они не могут обеспечить требуемую производительность и гибкость в облачной, многопользовательской, многоверсионной среде.

Архитектурный разрыв

Задача SourceCraft — предоставить IDE-as-a-Service. gopls для этого не подходит по фундаментальным причинам:

❌ Монолитная архитектура:

gopls совмещает анализ кода и обслуживание запросов. Для большого проекта (как Kubernetes) "холодный старт" и первоначальный анализ могут занимать десятки секунд. Пользователь в браузере не будет столько ждать.

❌ Ориентация на локальный state:

Сервер хранит состояние (кэши, AST) в памяти и локально на диске, что не подходит для распределенных систем.

❌ Работа с "живыми" файлами:

gopls заточен под работу с файлами на диске, которые меняются "здесь и сейчас". Он не умеет эффективно работать с абстракцией "код в коммите a1b2c3d4".

Наш подход: Не исправлять gopls, а спроектировать новую архитектуру, которую можно будет адаптировать под облачную IDE.

Цель и задачи

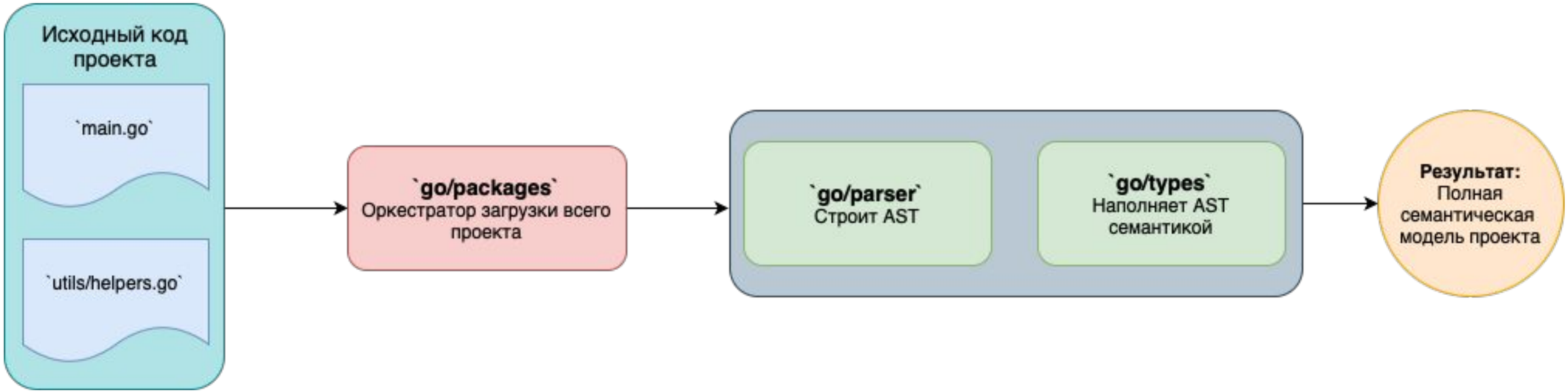
Цель: Разработать и исследовать прототип языкового сервиса для Go, решающего задачу мгновенной навигации по коду в многоверсионной среде платформы SourceCraft.

Задачи:

1. Спроектировать архитектуру и реализовать ядро анализа на основе пакетов Go и системы кэширования.
2. Реализовать расширение для VS Code и интегрировать сервер.
3. Реализовать обработчики LSP-запросов "definition" и "references".
4. Индексация по Git-коммитам.

Архитектура

Этап 1: Получение семантической модели кода
В основе нашего индексатора лежат стандартные компоненты компилятора Go. Это позволяет нам получать абсолютно точную информацию о коде, его структуре и связях.



- *go/packages* находит все файлы и зависимости проекта.
- *go/parser* и *go/types* совместно строят и анализируют код, создавая полную семантическую модель.

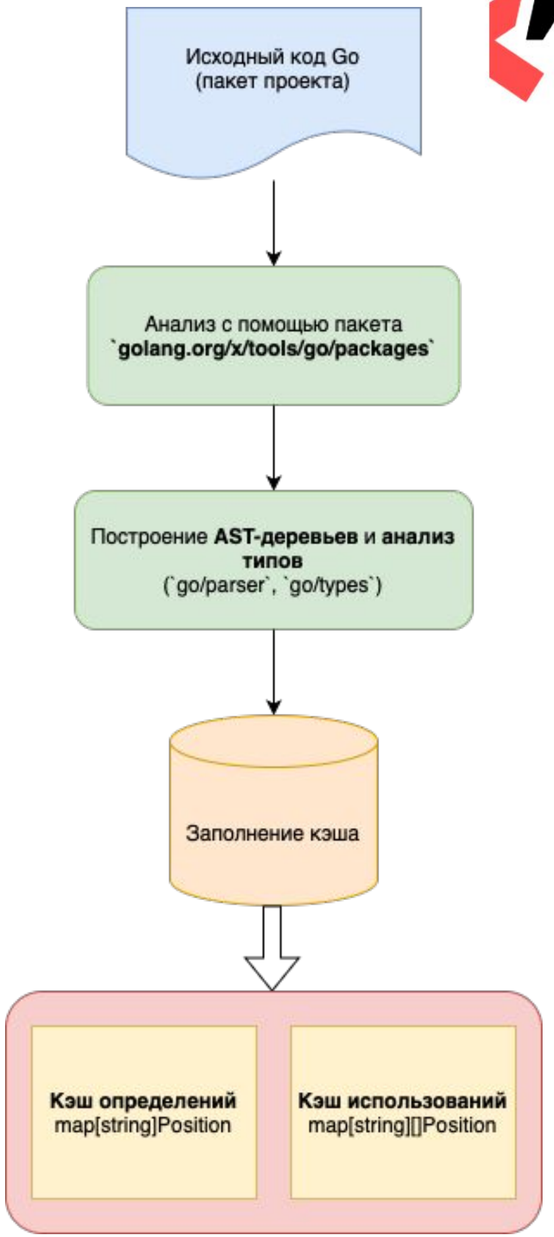
Результат этапа: Мы получаем богатую, но сложную для прямого использования структуру данных, которая содержит всю информацию о типах, объектах и их связях в проекте.

Архитектура

Этап 2: Трансформация модели в поисковый индекс
Хранить и запрашивать полную семантическую модель неэффективно. Поэтому мы трансформируем её в простую и быструю структуру данных, оптимизированную для навигационных запросов.

Извлечение данных: Мы итерируемся по семантической модели и извлекаем только два типа информации: где каждый символ объявлен и где он используется.
Заполнение кэша: Эта информация сохраняется в наш собственный индекс (кэш).

Структура индекса: По сути, это две хеш-таблицы (maps), но есть реализация и на БОР(unstable):
Кэш определений: Позволяет по уникальному ID символа мгновенно найти его место объявления (Position).
Кэш использований: Позволяет по тому же ID символа получить полный список всех мест его использования ([]Position).



Архитектура

Этап 2: Трансформация модели в поисковый индекс
Хранить и запрашивать полную семантическую модель неэффективно. Поэтому мы трансформируем её в простую и быструю структуру данных, оптимизированную для навигационных запросов.

Извлечение данных: Мы итерируемся по семантической модели и извлекаем только два типа информации: где каждый символ объявлен и где он используется.
Заполнение кэша: Эта информация сохраняется в наш собственный индекс (кэш).

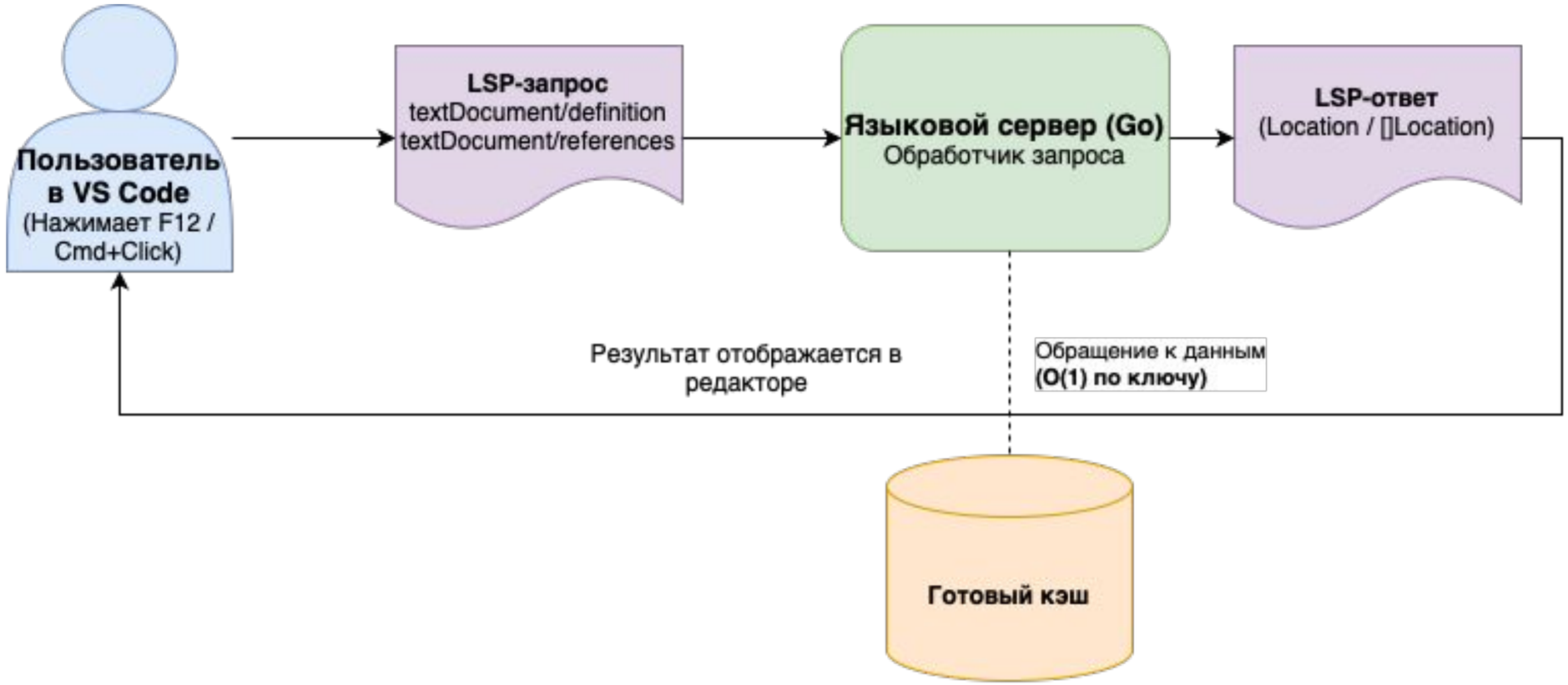
Структура индекса: По сути, это две хеш-таблицы (maps), но есть реализация и на БОР(unstable):
Кэш определений: Позволяет по уникальному ID символа мгновенно найти его место объявления (Position).
Кэш использований: Позволяет по тому же ID символа получить полный список всех мест его использования ([]Position).

```
type Declaration struct {
    ID       string
    Name     string
    Type     string
    Position string
    Package  string
}

type Usage struct {
    ID          string
    Position    string
    DeclID      string
    EndPosition string
}
```

Работа языкового сервера: от клика до результата

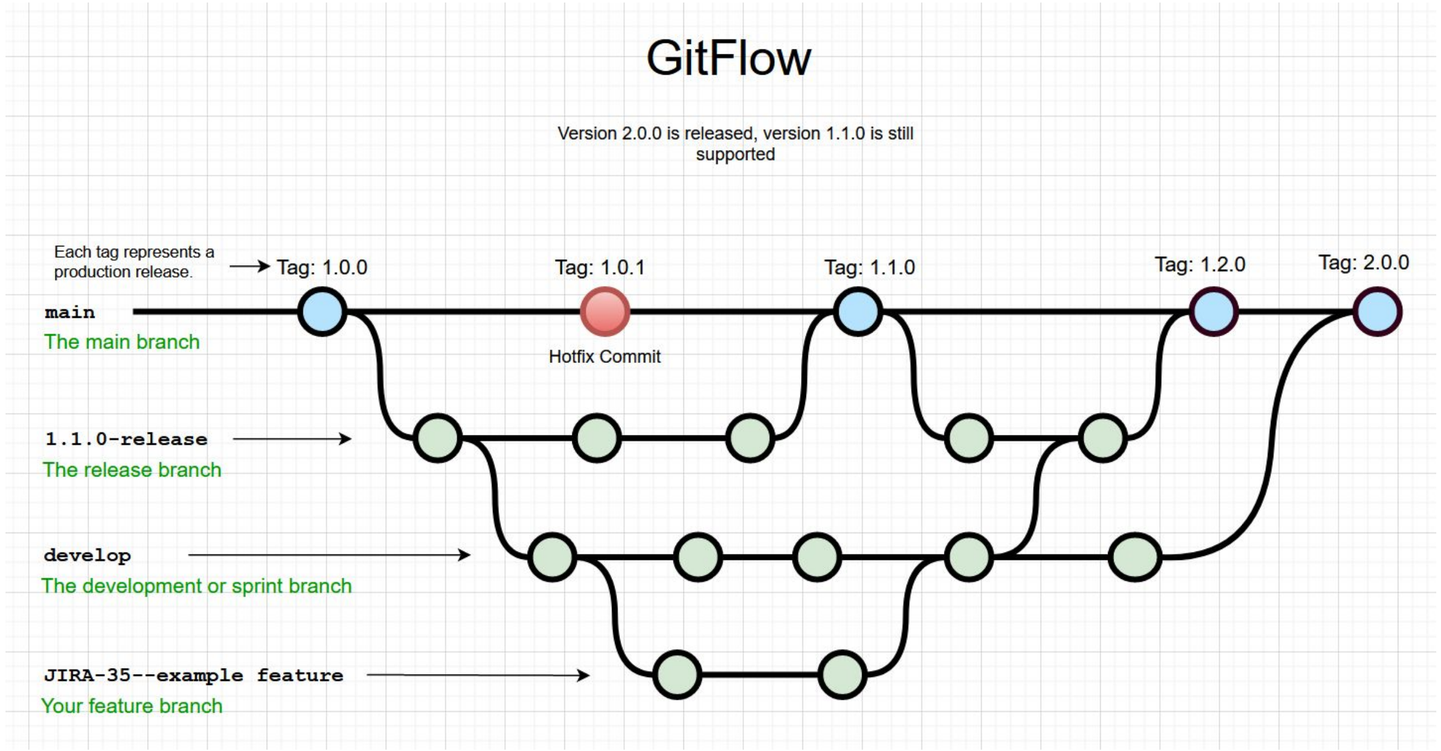
Второй компонент нашей архитектуры — легковесный LSP-сервер. По однажды проиндексированному коду сервер мгновенно отвечает на запросы, используя готовые данные.



Индексация каждого коммита

Платформы для совместной разработки, такие как SourceCraft, требуют навигации не только по актуальной версии кода, но и по всей его истории. С помощью библиотеки go-git был реализован проход по коммитам и формирование индекса для каждой версии проекта.

Реализованный процесс: Наш “сервис-индексатор” автоматизирует этот процесс, и позволяет получить индекс для каждого коммита, однако данная функциональность пока что не связана с LS.



Результаты

- Было разработано и реализовано построение индекса.
- Написан языковой сервер и обработчики для событий инициализации, обновления файлов, поиска деклараций и использований.
- Протестирована работоспособность на таких репозиториях как docker compose, kubernetes.
- Реализована индексация по git-коммитам.
- Рассмотрены различные подходы для будущей реализации инкрементальности.
- Написано около 3000 строк кода на Go.

Планы

1) Далее планируется внедрить инкрементальное обновление кэша при изменении файлов проекта.

2) Также планируется работа языкового сервера не только в текущей версии проекта, но и со всеми его версиями которые отражены в git-графе.

Спасибо за внимание!