ТРЕК PostgreSQL

КОМАНДА The_Last_Siberia

Алёна Куликова

КАПИТАН

ALENA KULIKOVA

ПОДДЕРЖКА КАПИТАНА

Задача

Разработать решение для мониторинга и анализа SQL-запросов к PostgreSQL, которое:

— оценивает «стоимость» запроса до его выполнения;

— дает рекомендации по оптимизации как самого запроса, так и базы данных;

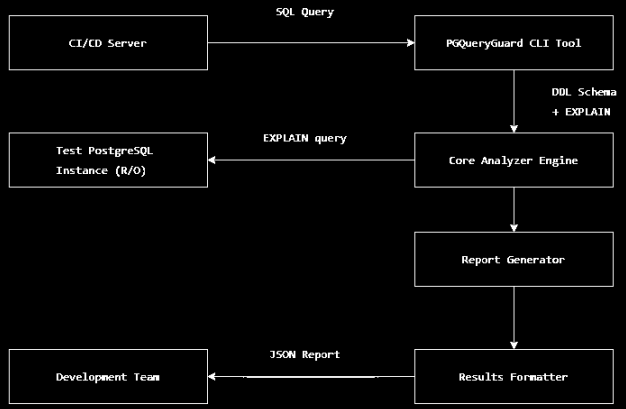
— помогает избежать проблем с производительностью.

Цель

Умный инструмент для аналитиков, DBA и разработчиков, который предупреждает о рисках и сокращает время ответа БД

Архитектура решения

- **CI/CD Server** - Точка входа в систему
  - Инициирует процесс анализа при изменениях в SQL-файлах
  - Предоставляет тестовое окружение с PostgreSQL
  - Запускает PGQueryGuard CLI Tool
- **PGQueryGuard CLI Tool** - Основной интерфейс взаимодействия с системой
  - Парсинг входных параметров
  - Координация работы между компонентами
  - Форматирование вывода для разных форматов
- **Core Analyzer Engine** - Интеллектуальное ядро системы
  - Генерация и выполнение EXPLAIN запросов
  - Парсинг JSON-ответов от PostgreSQL
  - Классификация рекомендаций по приоритетам
- **Test PostgreSQL Instance** - Контролируемая тестовая среда для анализа
  - Хранение актуальной схемы БД
  - Выполнение EXPLAIN запросов
  - Предоставление статистики и метрик
- **Report Generator** - Преобразование сырых данных в структурированные отчеты
  - Расчет общего score качества запроса
  - Формирование структурированного JSON-отчета
- **Results Formatter** - Адаптация результатов для конечных потребителей
- **Development Team** - Конечные потребители системы



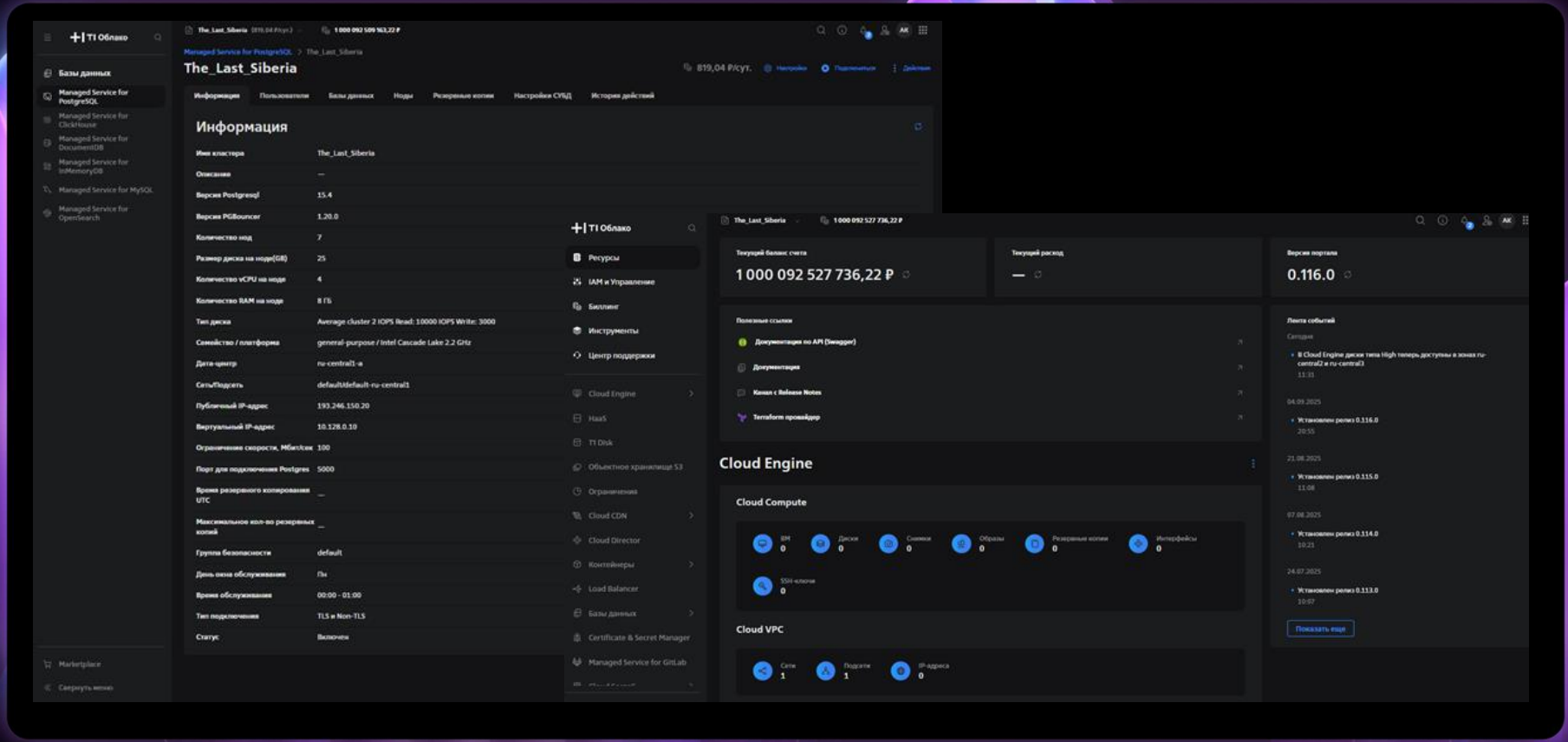
Технологический стек

- Язык
  Python 3.10+
- База данных
  PostgreSQL 15+
- Основные зависимости
  1. psycopg2-binary
  2. pydantic
  3. rich
  4. typer
  5. pyyaml

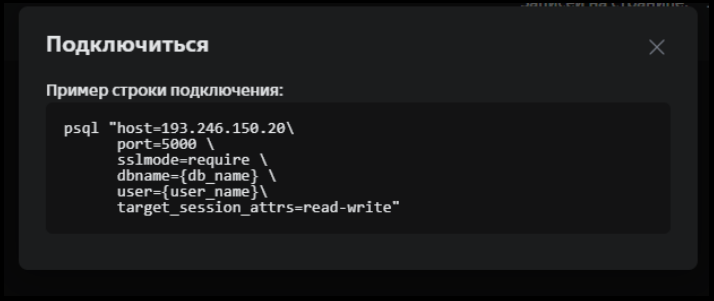
Технические трудности

- К сожалению присоединилась буквально за неделю до окончания
- К сожалению, доступ к Т1 Облако, получила за сутки, чтобы что-то настроить и реализовать адекватное

Реализация - Поднятие всех ресурсов

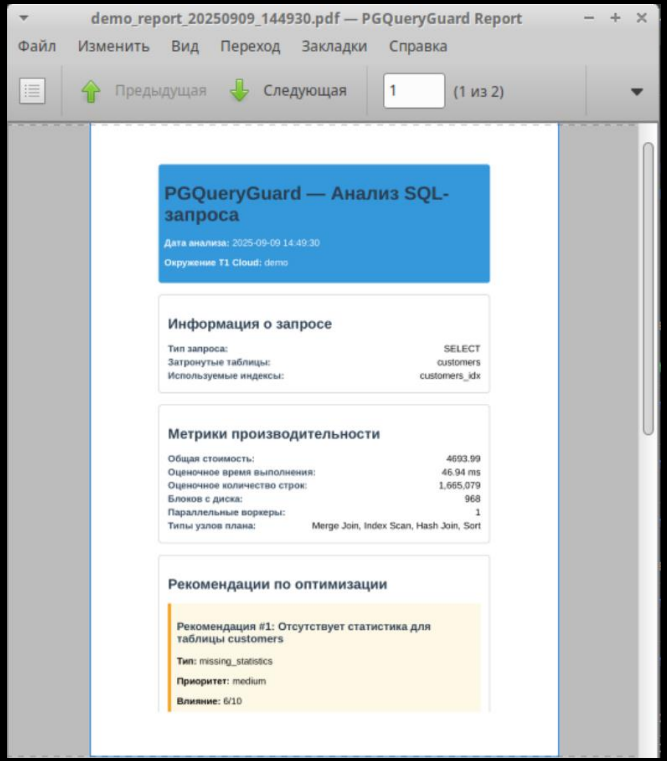


Реализация - Поднятие всех ресурсов



Реализация – Запуск демо (генерация отчета)

```
(.venv) alyona@alyona-virtualbox:~/PycharmProjects/PythonProjectha
caton$ python -m pgqueryguard.main
PDF-отчет успешно сохранен: output/demo_report_20250909_144930.pdf
```



Реализация – Описание главного кода

Устанавливает соединение с PostgreSQL и сохраняет информацию о сервере

- Подключается к БД с таймаутом 10 сек.
- Устанавливает application_name для мониторинга на стороне сервера.
- Автокоммит включён — так как мы только читаем планы (EXPLAIN).
- Выполняет запрос SELECT version(), current_database() — чтобы сохранить метаданные для отчёта.

```
33      def connect(self):  1 usage
34          try:
35              self.connection = psycopg2.connect(
36                  self.dsn,
37                  connect_timeout=10,
38                  application_name="pgqueryguard-t1-cloud"
39              )
40              self.connection.autocommit = True
41
42              with self.connection.cursor() as cur:
43                  cur.execute("SELECT version(), current_database()")
44                  version_info, db_name = cur.fetchone()
45                  self.server_version = version_info
46                  self.db_name = db_name
47
48          except psycopg2.Error as e:
49              raise
```

Реализация – Описание главного кода

Получает план выполнения запроса в формате JSON

- Сначала пытается выполнить EXPLAIN (FORMAT JSON, VERBOSE, SETTINGS, BUFFERS) — максимально детальный план.
- Если возникает ошибка (например, нет прав на BUFFERS) — fallback на EXPLAIN (FORMAT JSON).
- Парсит JSON и возвращает словарь.

```python
    def get_explain_plan(self, query: str) -> Dict[str, Any]:  1 usage
        with self.connection.cursor() as cur:
            try:
                explain_query = f"EXPLAIN (FORMAT JSON, VERBOSE, SETTINGS, BUFFERS) {query}"
                cur.execute(explain_query)
                result = cur.fetchone()

                if result and result[0]:
                    plan_data = json.loads(result[0])
                    return plan_data
                else:
                    raise Exception("Пустой результат EXPLAIN")

            except psycopg2.Error:
                try:
                    explain_query = f"EXPLAIN (FORMAT JSON) {query}"
                    cur.execute(explain_query)
                    result = cur.fetchone()
                    if result and result[0]:
                        plan_data = json.loads(result[0])
                        return plan_data
                except psycopg2.Error as e:
                    raise Exception(f"Не удалось получить план выполнения: {e}")
```

Реализация – Описание главного кода

Извлекает числовые метрики производительности из плана выполнения и возвращает объект QueryMetric

total_cost — общая стоимость запроса по оценке планировщика.
planning_time — время планирования (если есть).
max_execution_time — оценка времени выполнения
shared_hit_blocks — блоки, прочитанные из кэша.
shared_read_blocks — блоки, прочитанные с диска.
plan_width — средняя ширина строки результата
total_rows — оценочное количество строк.
node_types — список уникальных типов узлов плана
startup_cost — стоимость до начала возврата первой строки.
total_workers — общее количество воркеров
parallel_workers — количество запущенных параллельных воркеров.

```python
    def extract_metrics(self, plan: Dict[str, Any]) -> QueryMetric:  1 usage
        total_plan = plan[0]['Plan']

        metrics = QueryMetric(
            total_cost=total_plan['Total Cost'],
            planning_time=total_plan.get('Planning Time'),
            max_execution_time=self.estimate_execution_time(total_plan['Total Cost']),
            shared_hit_blocks=total_plan.get('Shared Hit Blocks', 0),
            shared_read_blocks=total_plan.get('Shared Read Blocks', 0),
            plan_width=total_plan['Plan Width'],
            total_rows=total_plan['Plan Rows'],
            node_types=self.extract_node_types(total_plan),
            startup_cost=total_plan.get('Startup Cost', 0),
            total_workers=total_plan.get('Workers', 0),
            parallel_workers=total_plan.get('Workers Launched', 0)
        )

        return metrics
```

Реализация – Описание главного кода

Рекурсивно обходит дерево плана выполнения и собирает уникальные типы узлов

```
94      def extract_node_types(self, plan_node: Dict[str, Any]) -> List[str]:  1 usage
95          node_types = []
96
97          def _extract_nodes(node):
98              if 'Node Type' in node:
99                  node_types.append(node['Node Type'])
100             if 'Plans' in node:
101                 for child in node['Plans']:
102                     _extract_nodes(child)
103
104         _extract_nodes(plan_node)
105         return list(set(node_types))
```

Заглушка для анализа структуры плана

```
107     def analyze_plan_structure(self, plan: Dict[str, Any]) -> None:  1 usage
108         total_plan = plan[0]['Plan']
109
110         def build_tree(node, parent_tree=None):
111             if 'Plans' in node:
112                 for child in node['Plans']:
113                     build_tree(child, parent_tree)
114
115         build_tree(total_plan)
```

Реализация – Описание главного кода

Преобразует "стоимость" запроса в оценочное время выполнения

```python
def estimate_execution_time(self, total_cost: float) -> float:  1 usage
    return total_cost * 0.01
```

Генерирует список предупреждений на основе анализа плана и текста запроса

```python
def extract_indexes_used(self, plan: Dict[str, Any]) -> List[str]:  1 usage
    indexes = []

    def _find_indexes(node):
        if node.get('Node Type') == 'Index Scan':
            index_name = node.get('Index Name', 'unknown')
            relation_name = node.get('Relation Name', 'unknown')
            indexes.append(f"{relation_name}({index_name})")

        if 'Plans' in node:
            for child in node['Plans']:
                _find_indexes(child)

    _find_indexes(plan[0]['Plan'])
    return indexes
```

Реализация – Описание главного кода

Генерирует список рекомендаций по оптимизации запроса с учетом особенностей T1 Cloud

Каждая рекомендация — объект Recommendation с описанием, приоритетом, оценкой улучшения, действием и сервисом T1 Cloud

Таблица типов рекомендации

```python
def generate_t1_recommendations(self, plan: Dict[str, Any], query: str) -> List[Recommendation]:  1 usage
    recommendations = []
    total_plan = plan[0]['Plan']

    seq_scans = find_plan_nodes(total_plan, node_type: 'Seq Scan')
    for scan in seq_scans:
        if scan['Plan Rows'] > 10000 and 'Filter' in scan:
            table_name = scan.get('Relation Name', 'unknown')
            rec = Recommendation(
                type="missing_index",
                description=f"Полное сканирование большой таблицы {table_name} ({scan['Plan Rows']:,} строк)",
                priority=Priority.HIGH,
                estimated_improvement="Ускорение на 80-95%",
                suggested_action=f"Создать индекс на {table_name}({extract_filter_columns(scan)})",
                affected_components=[table_name],
                t1_service=T1CloudService.POSTGRESQL,
                impact_score=9
            )
            recommendations.append(rec)
```

	Standard	Standard	Standard	Standard
1	Тип	Условие	Приоритет	Описание
2	missing_index	Seq Scan + Filter + >10K строк	HIGH	Создать индекс на полях фильтра
3	disk_sort	Sort Method = external	MEDIUM	Увеличить work_mem или оптимизировать ORDER BY
4	inefficient_join	Nested Loop + >1K строк	MEDIUM	Рассмотреть Hash Join"

Реализация – Описание главного кода

Рассчитывает общий score запроса на основе метрик и рекомендаций

Стоимость запроса

10000 → -30 5000 → -20 1000 → -10

Дисковые чтения

1000 блоков → -25 500 блоков → -15

Рекомендации

HIGH → -30 MEDIUM → -15

```python
def calculate_score(self, metrics: QueryMetric, recommendations: List[Recommendation]) -> int:  1 usage
    base_score = 100

    if metrics.total_cost > 10000:
        base_score -= 30
    elif metrics.total_cost > 5000:
        base_score -= 20
    elif metrics.total_cost > 1000:
        base_score -= 10

    if metrics.shared_read_blocks > 1000:
        base_score -= 25
    elif metrics.shared_read_blocks > 500:
        base_score -= 15

    for rec in recommendations:
        if rec.priority == Priority.HIGH:
            base_score -= 30
        elif rec.priority == Priority.MEDIUM:
            base_score -= 15

    return max(0, min(100, base_score))
```

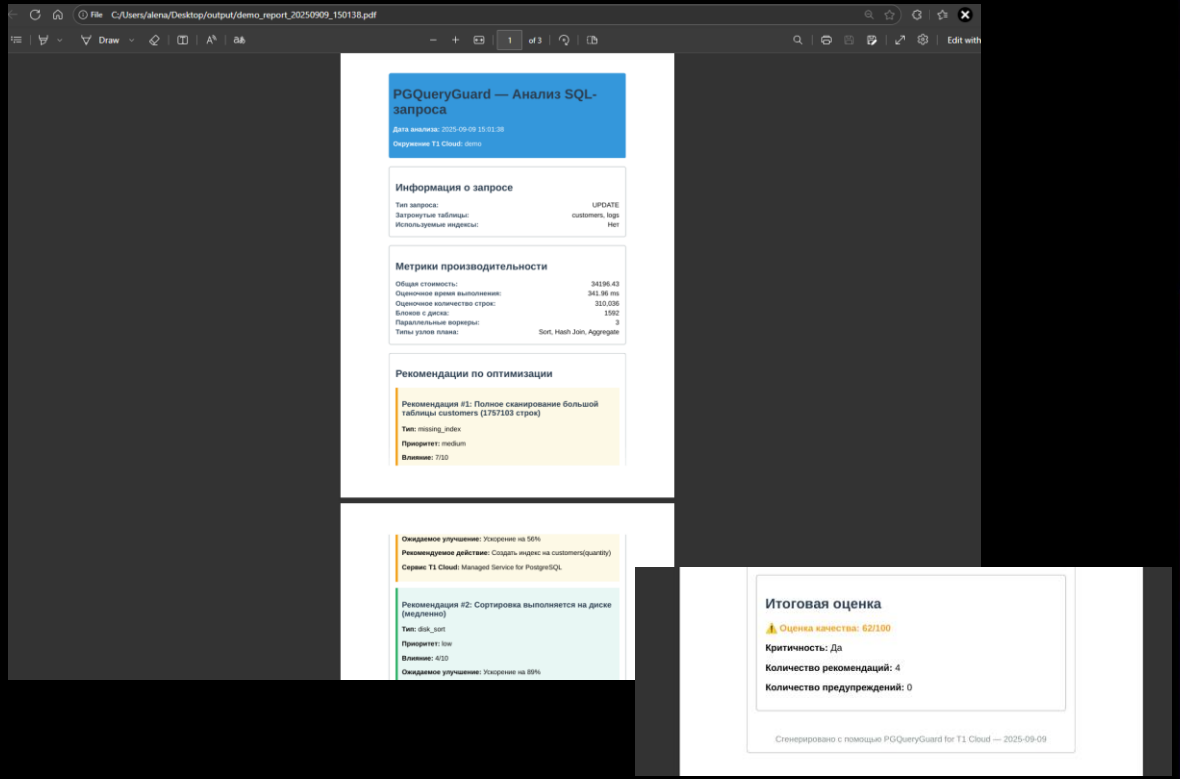
Реализация – Описание главного кода

выполняет полный анализ SQL-запроса и возвращает структурированный отчёт

```
225     def analyze_query(self, query: str) -> AnalysisReport:  1 usage
226         query_info = extract_query_info(query)
227         plan = self.get_explain_plan(query)
228         if self.verbose:
229             self.analyze_plan_structure(plan)
230         metrics = self.extract_metrics(plan)
231         recommendations = self.generate_t1_recommendations(plan, query)
232         report = AnalysisReport(
233             query=query,
234             metrics=metrics,
235             recommendations=recommendations,
236             is_critical=any(r.priority == Priority.HIGH for r in recommendations),
237             score=self.calculate_score(metrics, recommendations),
238             t1_environment=self.t1_environment,
239             query_type=query_info['type'],
240             tables_affected=query_info['tables'],
241             indexes_used=self.extract_indexes_used(plan),
242             warnings=self.generate_warnings(plan, query)
243         )
244         return report
```

Реализация – Оценка отчета

Как можно заметить, отчет составляется информативно и предоставляет рекомендацию



Результаты

- **Разработано** решение для мониторинга и анализа SQL-запросов к PostgreSQL
- **Не смотря на тех. трудности** и отсутствие времени, имеется прототип проекта, который можно развить до автоматизации, что на данный момент является крайне важным:
  - Команды разработки получают проактивные рекомендации до попадания запроса в прод
  - Администраторы БД могут предотвращать инциденты производительности
  - Снижается нагрузка на кластеры PostgreSQL в T1 Cloud