Avito

Хранимки as a code

О докладчике

- Разработчик баз данных
- DBA
- Backend разработчик
- ...
- Software engineer

О докладчике

- Разработчик баз данных
- DBA
- Backend разработчик
- ...
- Software engineer

О докладе

О докладе

- О хранимых функциях и процедурах (хранимках)

О докладе

- О хранимых функциях и процедурах (хранимках)
- Когда и почему стоит писать хранимки

О докладе

- О хранимых функциях и процедурах (хранимках)
- Когда и почему стоит писать хранимки
- Когда и почему **не стоит** писать хранимки

О докладе

- О хранимых функциях и процедурах (хранимках)
- Когда и почему стоит писать хранимки
- Когда и почему **не стоит** писать хранимки
- Best practice по работе с хранимками

О докладе

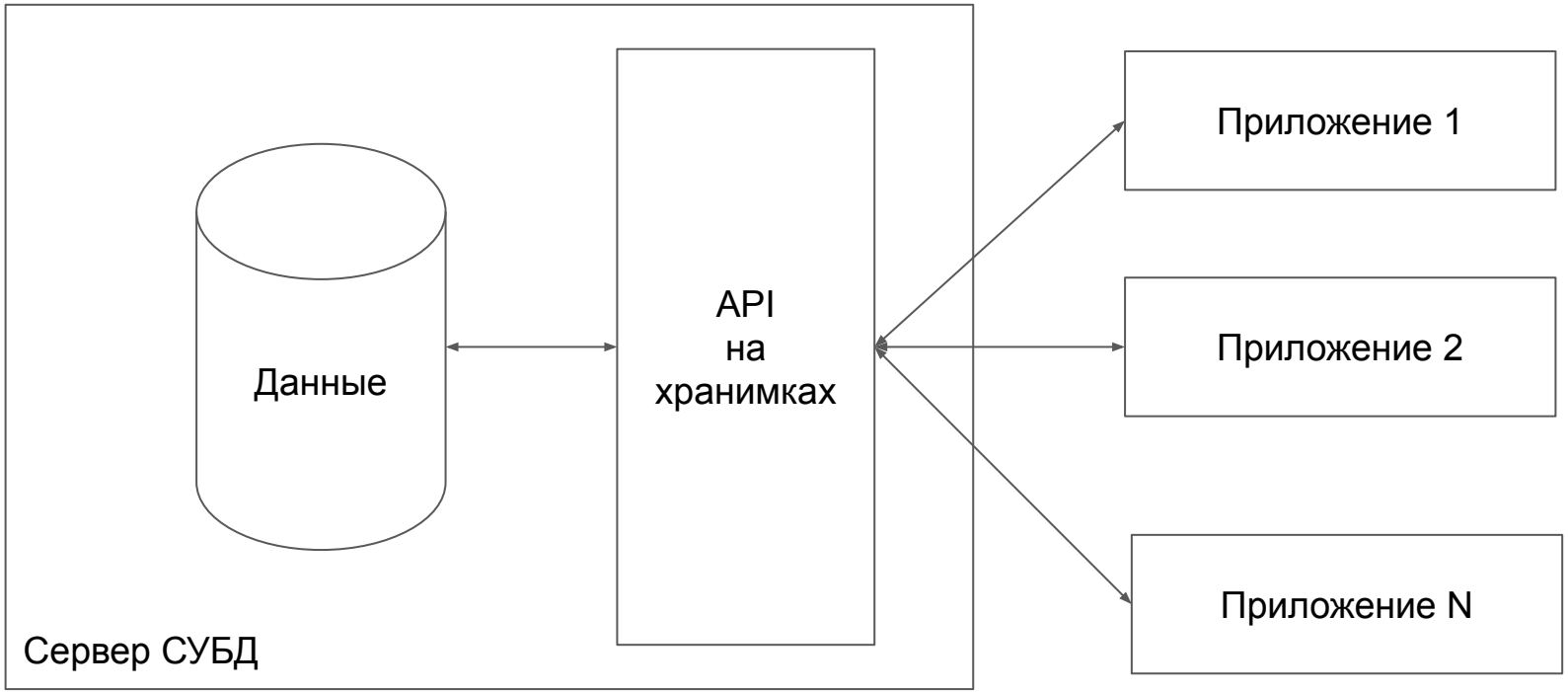
- О хранимых функциях и процедурах (хранимках)
- Когда и почему стоит писать хранимки
- Когда и почему **не стоит** писать хранимки
- Best practice по работе с хранимками
- Как доставлять до production

О докладе

- О хранимых функциях и процедурах (хранимках)
- Когда и почему стоит писать хранимки
- Когда и почему **не стоит** писать хранимки
- Best practice по работе с хранимками
- Как доставлять до production
- ...
- Много всего, поэтому без глубокого погружения

Когда и почему стоит хранимки

Дополнительный слой абстракции



Дополнительный слой абстракции

Дополнительный слой абстракции

- Переиспользование кода

Дополнительный слой абстракции

- Переиспользование кода
- Инкапсуляция
  - Прозрачное секционирование
  - Прозрачная нормализация и денормализация
  - Прозрачные миграции данных

Дополнительный слой абстракции

- Переиспользование кода
- Инкапсуляция
  - Прозрачное секционирование
  - Прозрачная нормализация и денормализация
  - Прозрачные миграции данных
- Разделение ответственностей и компетенций

Безопасность

Безопасность

- Ограничение возможных действий над данными

Безопасность

- Ограничение возможных действий над данными
- Сложная ролевая модель

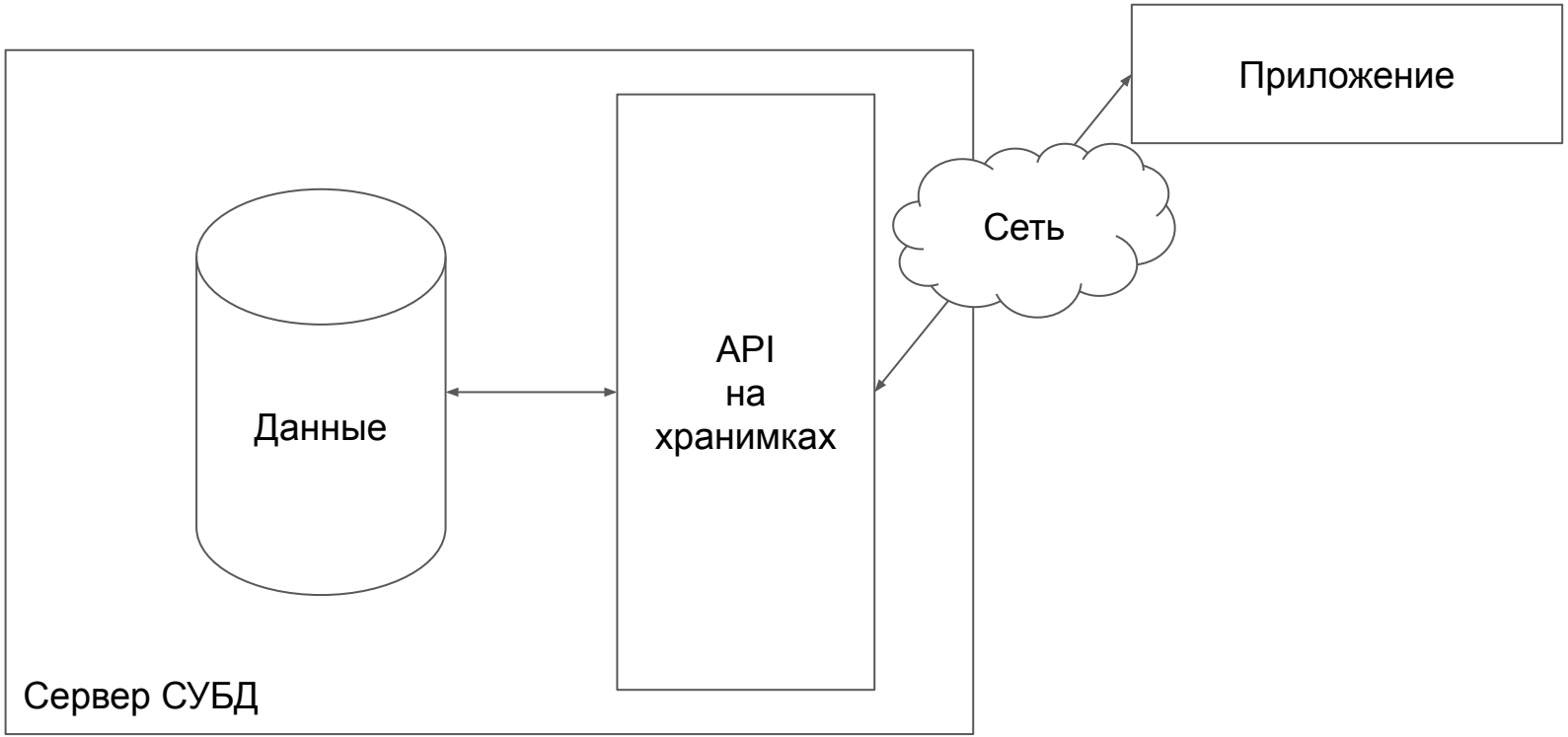
Безопасность

- Ограничение возможных действий над данными
- Сложная ролевая модель
- Аудит

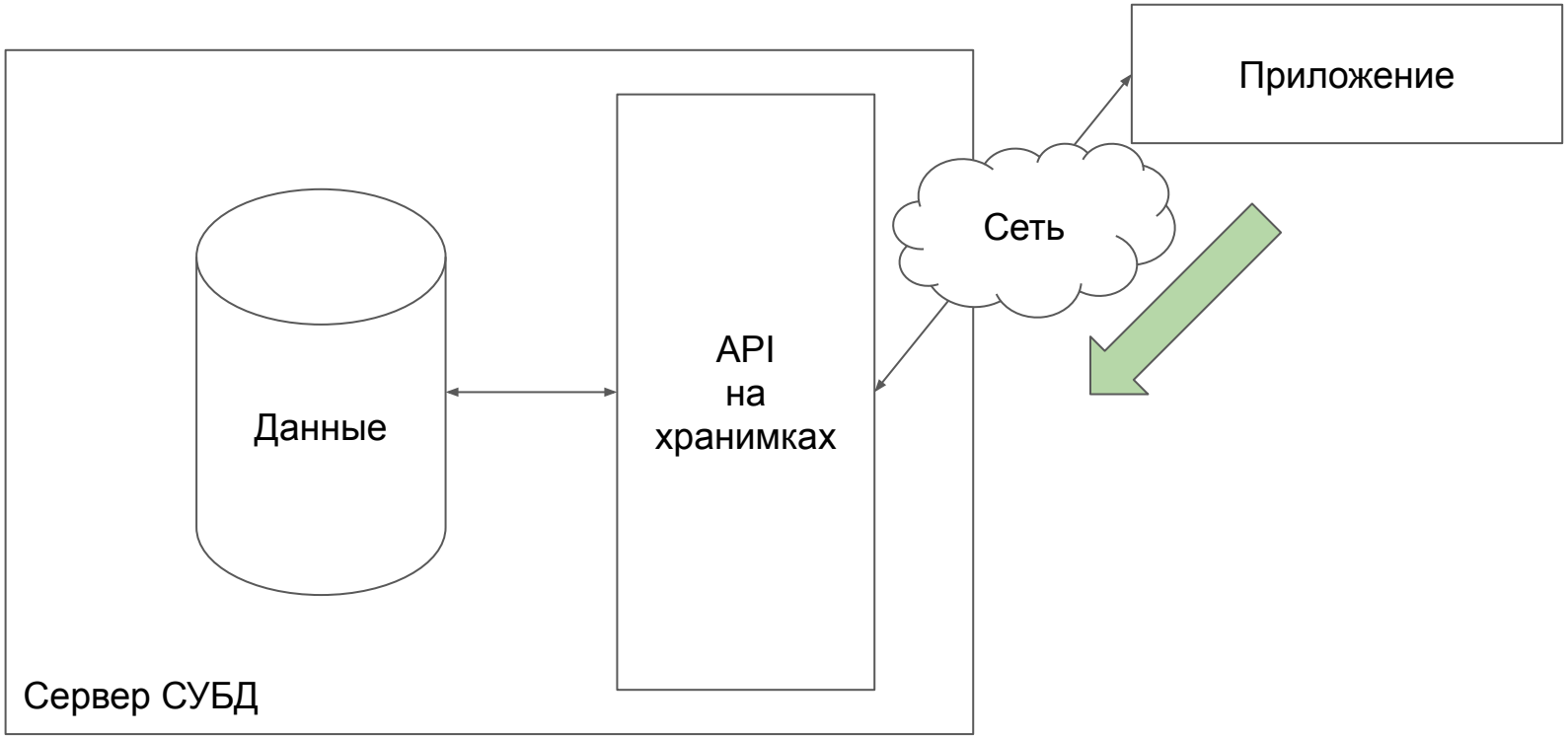
Безопасность

- Ограничение возможных действий над данными
- Сложная ролевая модель
- Аудит
- Административная функциональность без привилегий superuser через опцию security definer

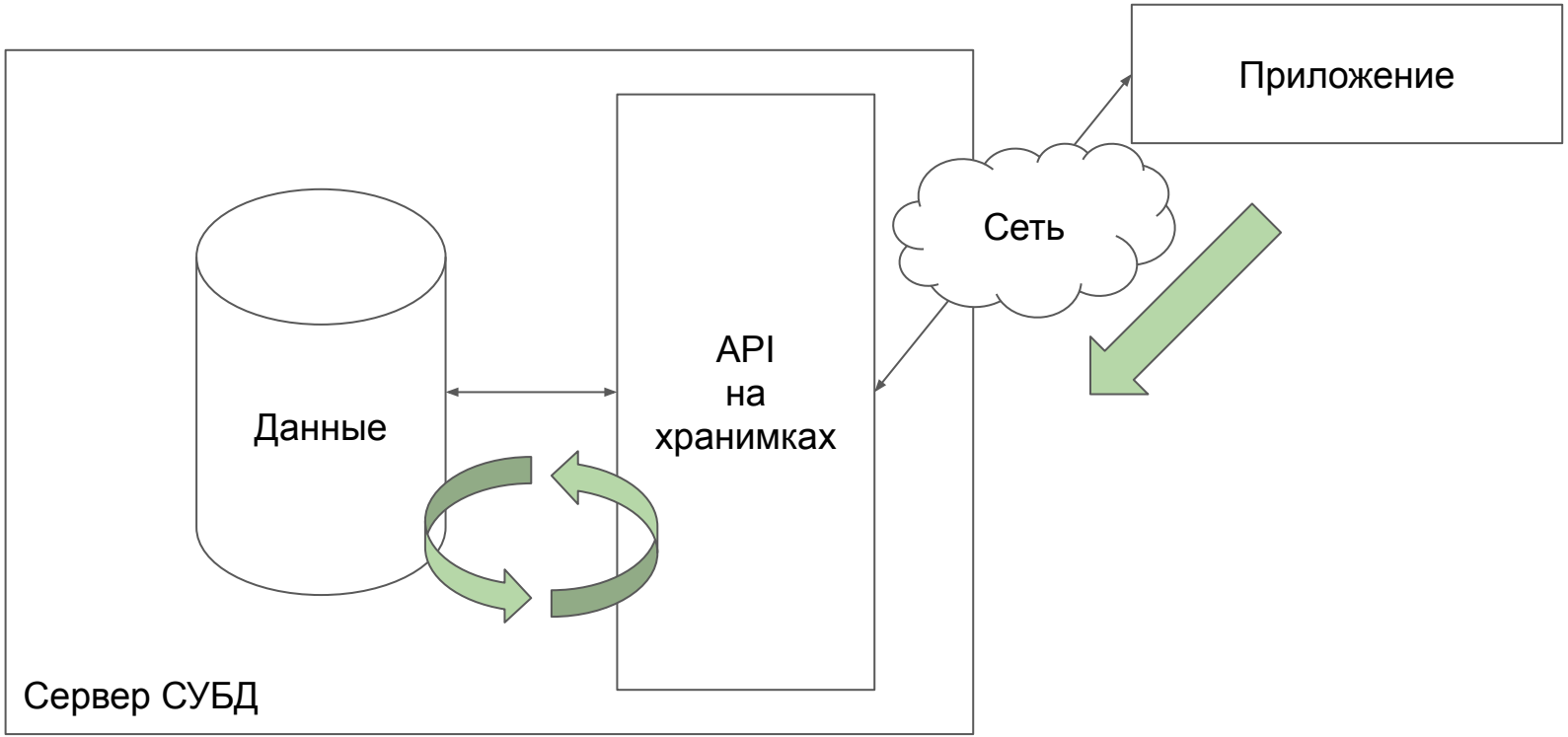
Локальность данных



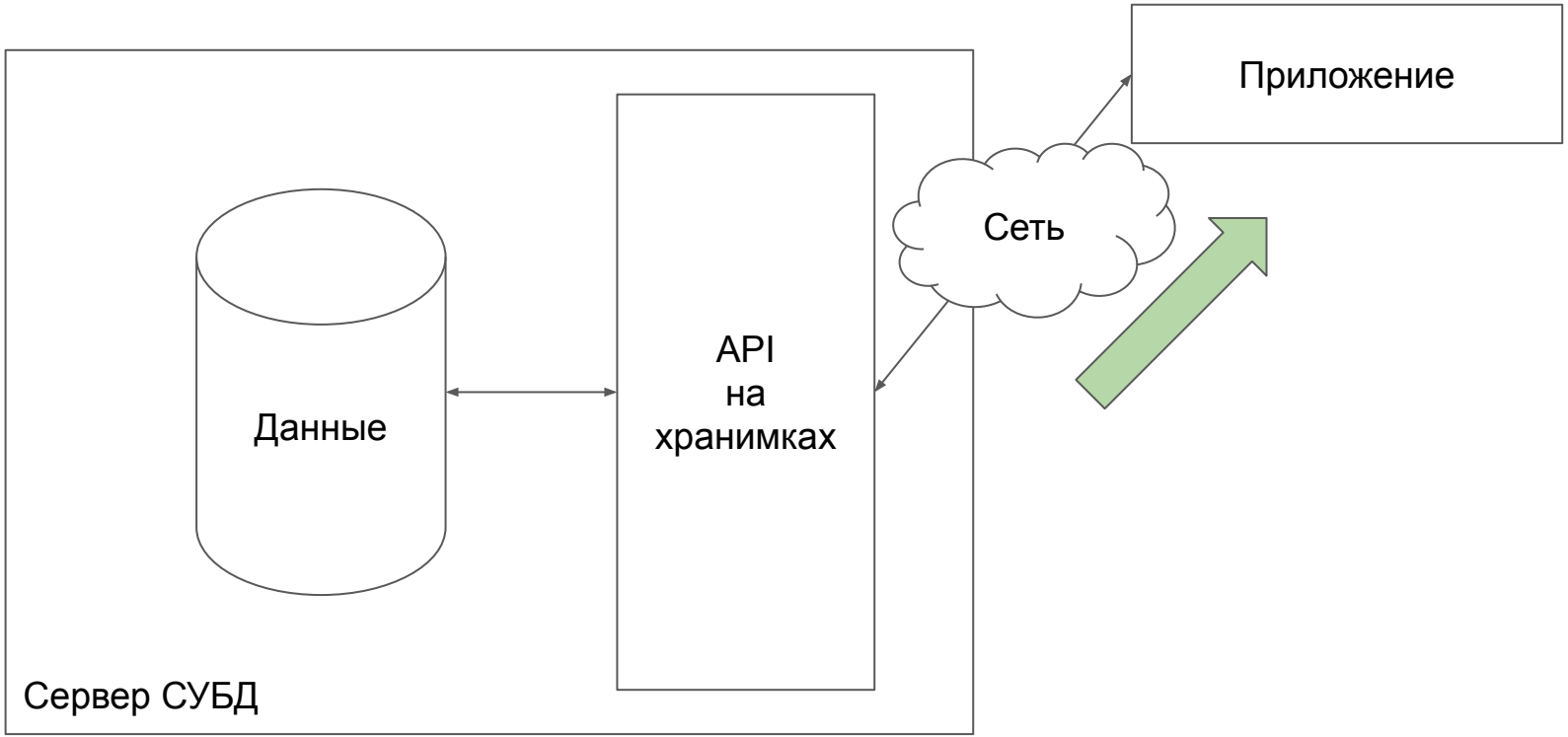
Локальность данных



Локальность данных



Локальность данных



Низкоуровневые трюки

Низкоуровневые трюки

- Триггерные функции

Низкоуровневые трюки

- Триггерные функции
- Всяческого рода “хинты”
  - work_mem
  - join_collapse_limit
  - enable_seqscan, enable_indexscan, …
  - enable_hashjoin, enable_mergejoin, ...

Низкоуровневые трюки

- Триггерные функции
- Всяческого рода “хинты”
  - work_mem
  - join_collapse_limit
  - enable_seqscan, enable_indexscan, …
  - enable_hashjoin, enable_mergejoin, ...
- Переменные транзакции и сессии через set_config/current_setting

Низкоуровневые трюки

- Триггерные функции
- Всяческого рода “хинты”
  - work_mem
  - join_collapse_limit
  - enable_seqscan, enable_indexscan, …
  - enable_hashjoin, enable_mergejoin, ...
- Переменные транзакции и сессии через set_config/current_setting
- Динамический SQL

Низкоуровневые трюки

- Триггерные функции
- Всяческого рода “хинты”
  - work_mem
  - join_collapse_limit
  - enable_seqscan, enable_indexscan, …
  - enable_hashjoin, enable_mergejoin, ...
- Переменные транзакции и сессии через set_config/current_setting
- Динамический SQL
- Доступ к ресурсам сервера СУБД (untrusted)

Когда и почему не стоит писать хранимки

Понятность, прозрачность и удобство

Понятность, прозрачность и удобство

- Бизнес-логика рассредоточена по нескольким слоям приложения

Понятность, прозрачность и удобство

- Бизнес-логика рассредоточена по нескольким слоям приложения
- Еще один язык программирования (компетенция)

Понятность, прозрачность и удобство

- Бизнес-логика рассредоточена по нескольким слоям приложения
- Еще один язык программирования (компетенция)
- Низкоуровневые и неочевидные оптимизации

Понятность, прозрачность и удобство

- Бизнес-логика рассредоточена по нескольким слоям приложения
- Еще один язык программирования (компетенция)
- Низкоуровневые и неочевидные оптимизации
- Сложность интеграции с ОО-моделью и всяческими ORM

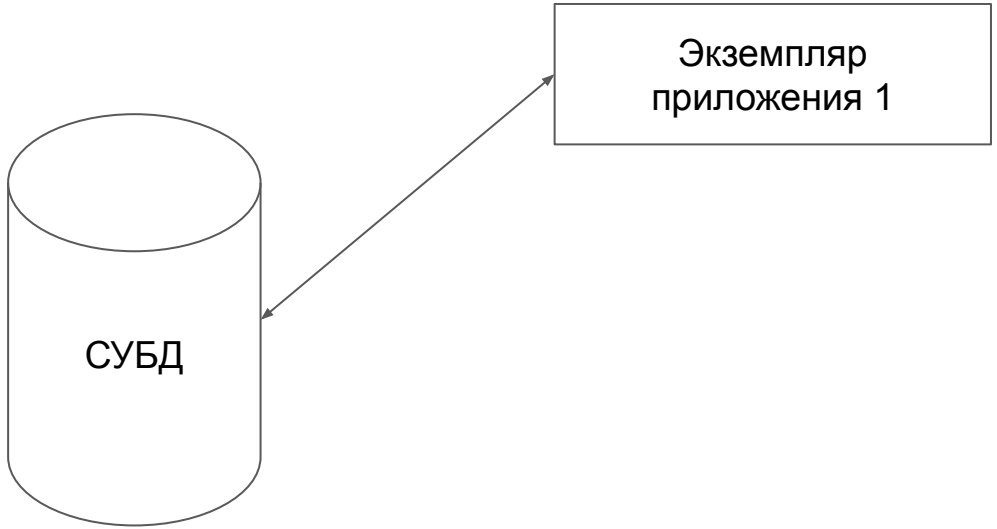
Понятность, прозрачность и удобство

- Бизнес-логика рассредоточена по нескольким слоям приложения
- Еще один язык программирования (компетенция)
- Низкоуровневые и неочевидные оптимизации
- Сложность интеграции с ОО-моделью и всяческими ORM
- Зависимость от конкретной СУБД

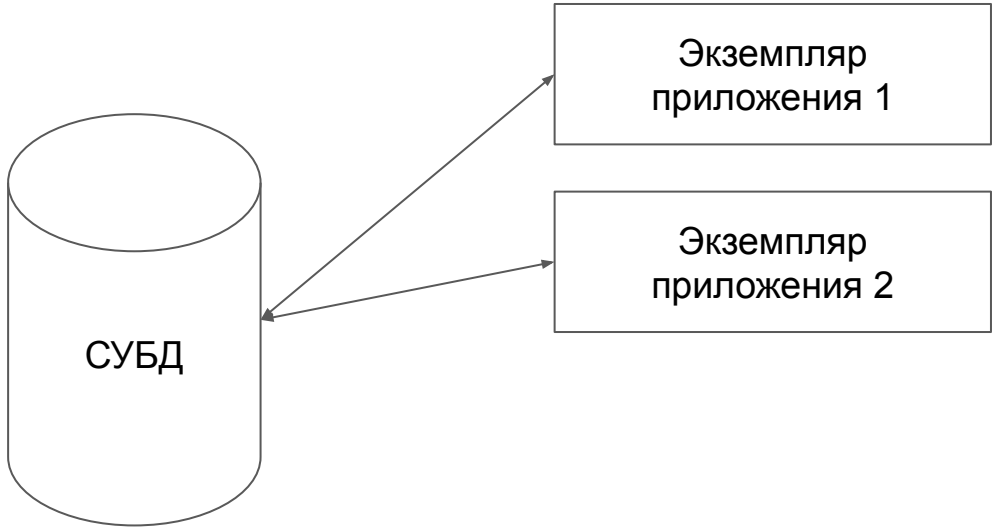
Понятность, прозрачность и удобство

- Бизнес-логика рассредоточена по нескольким слоям приложения
- Еще один язык программирования (компетенция)
- Низкоуровневые и неочевидные оптимизации
- Сложность интеграции с ОО-моделью и всяческими ORM
- Зависимость от конкретной СУБД
- Низкая скорость разработки и большая когнитивная нагрузка

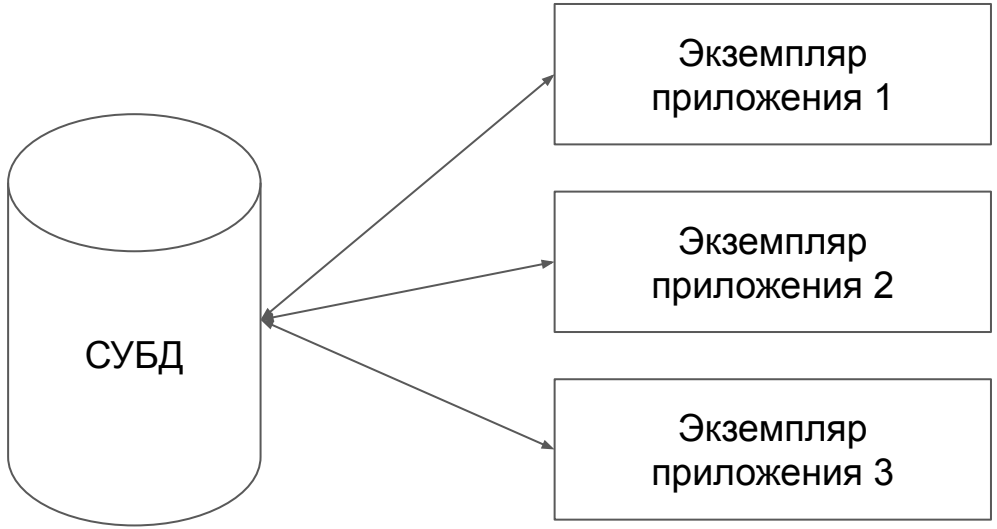
Масштабируемость



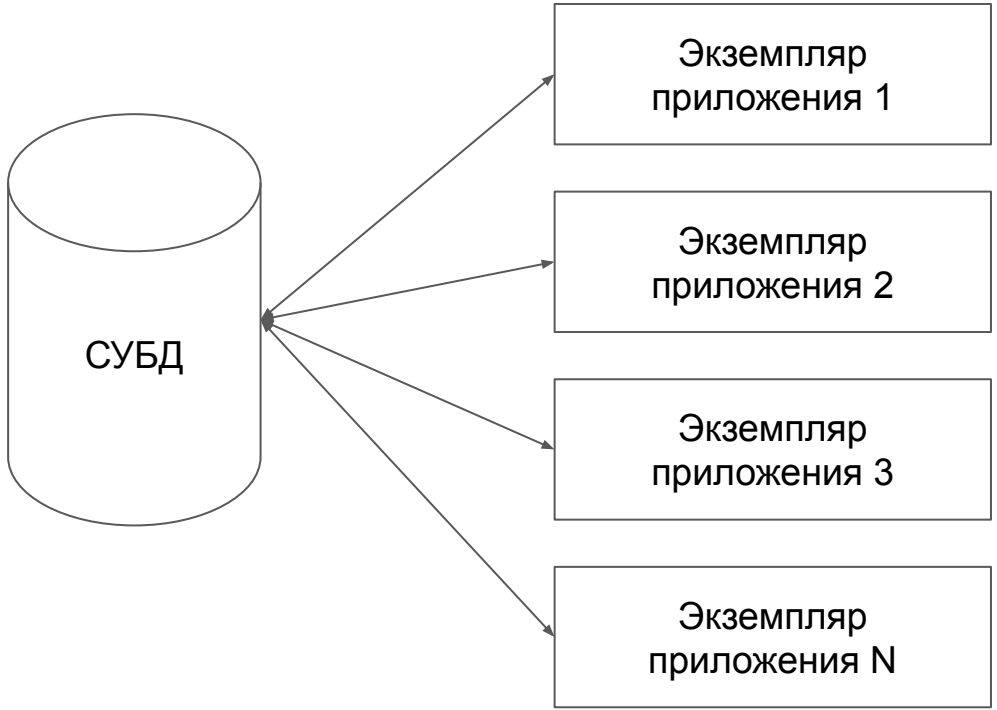
Масштабируемость



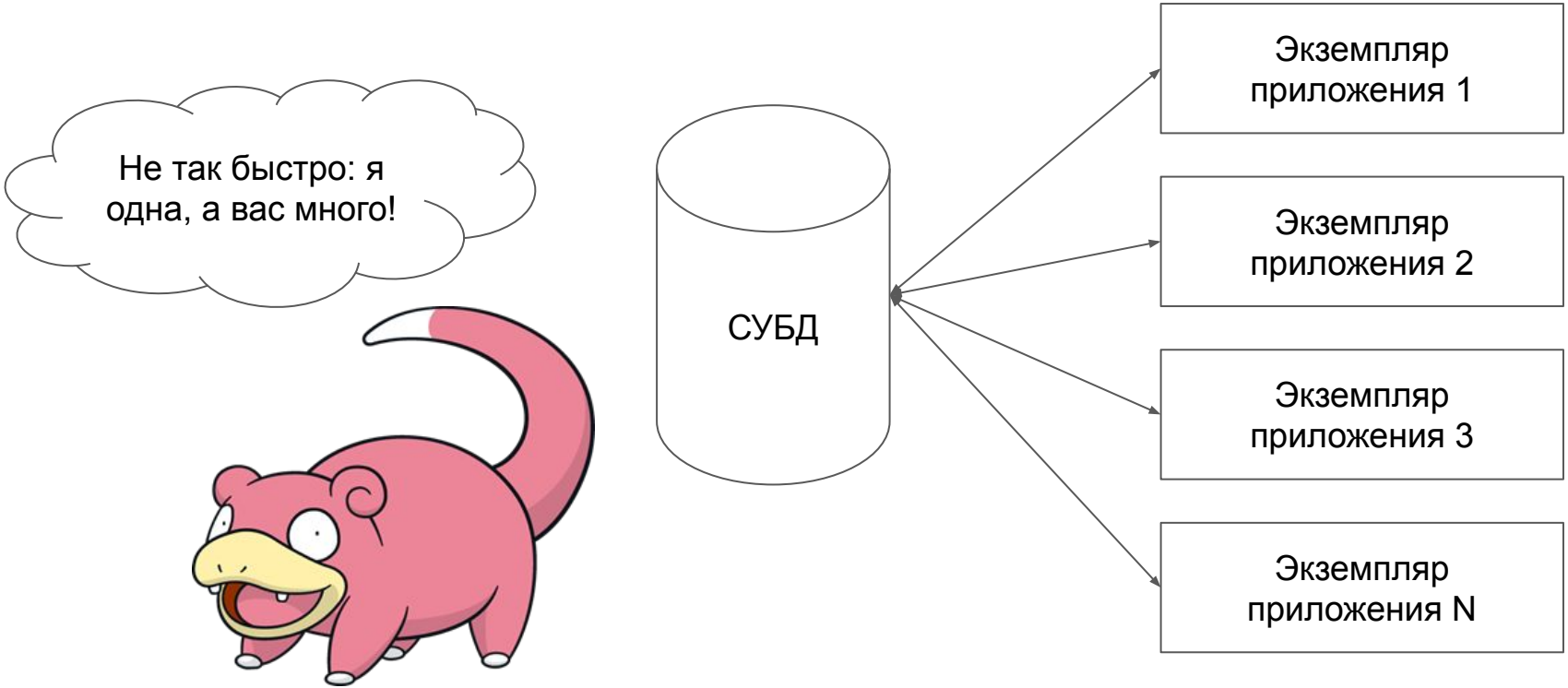
Масштабируемость



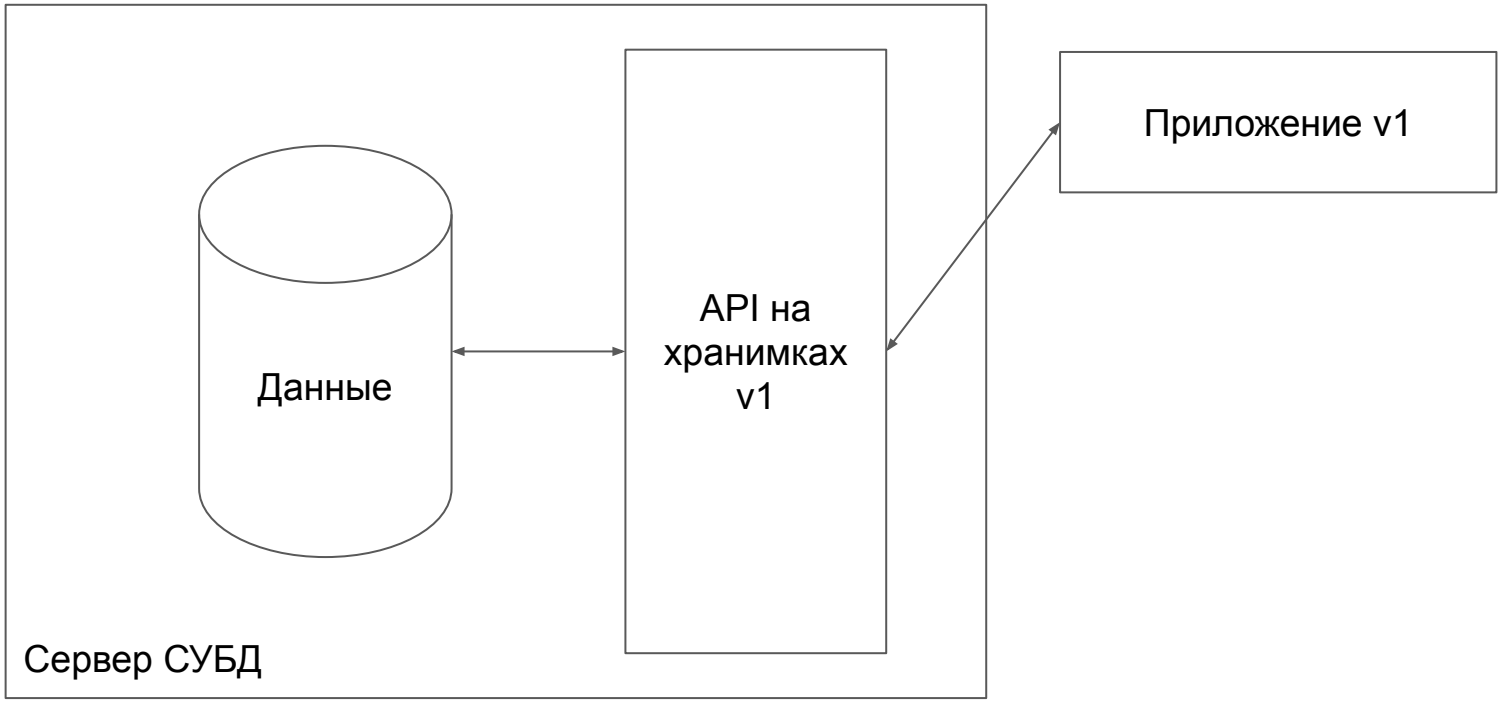
Масштабируемость



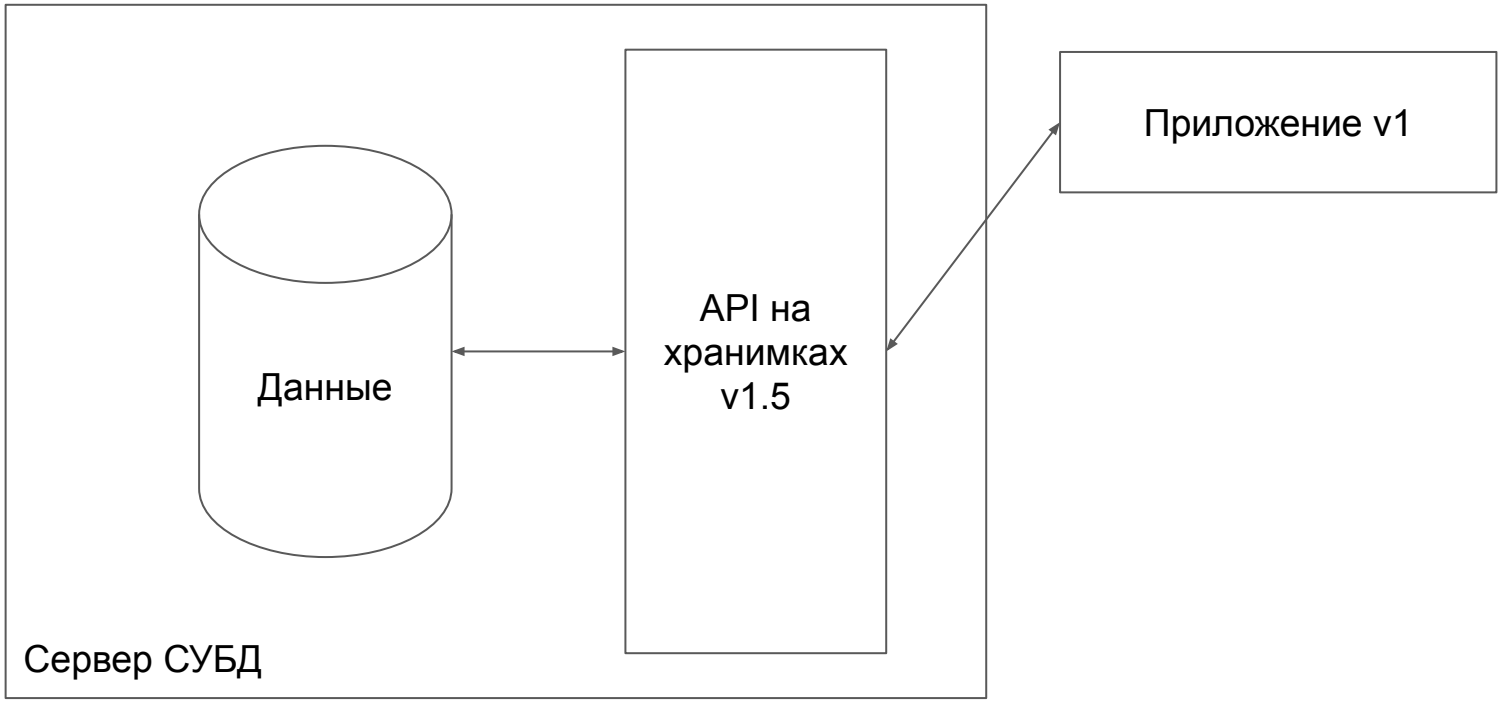
Масштабируемость



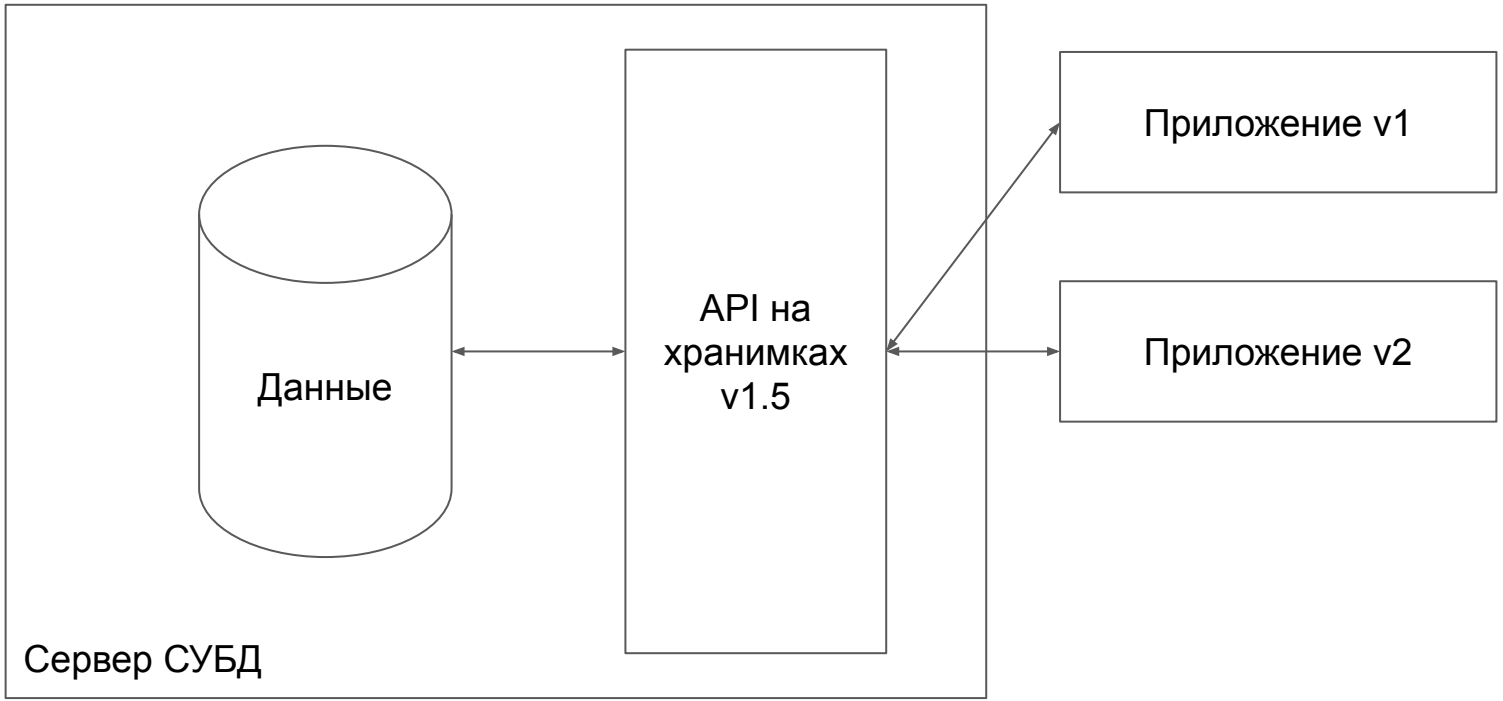
Неатомарный деплой



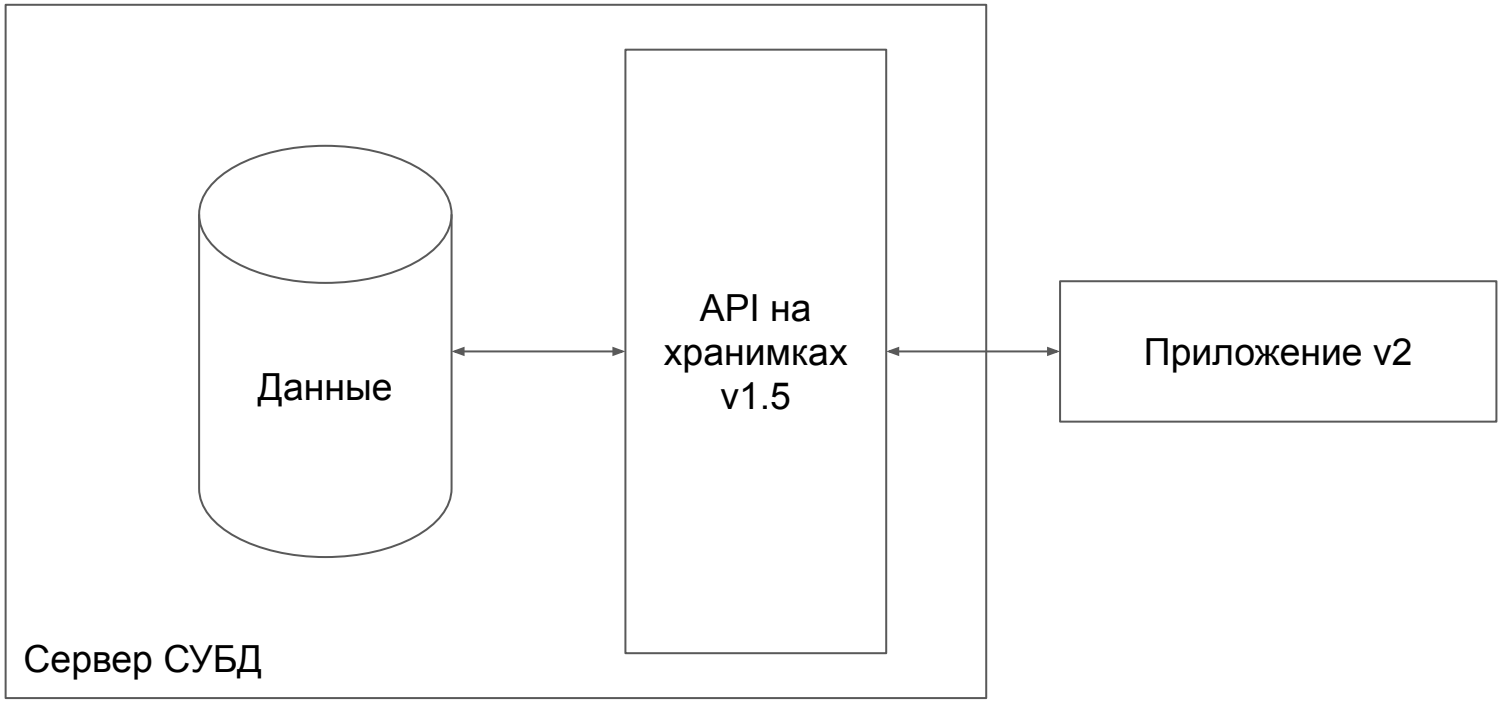
Неатомарный деплой



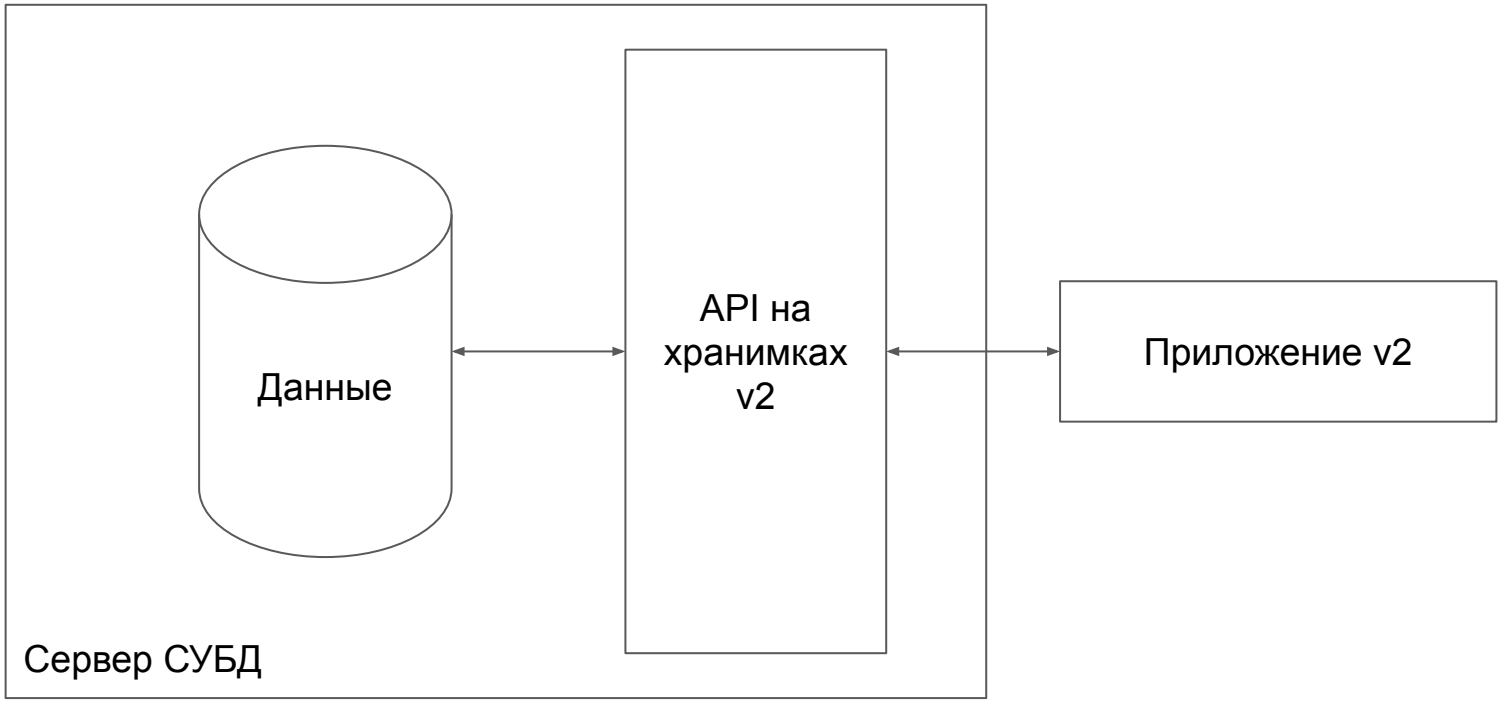
Неатомарный деплой



Неатомарный деплой



Неатомарный деплой



Языки хранимых процедур

Языки хранимых процедур

- не являются языками общего назначения

Языки хранимых процедур

- не являются языками общего назначения
- имеют низкую выразительность

Языки хранимых процедур

- не являются языками общего назначения
- имеют низкую выразительность
- имеют сомнительную производительность

Языки хранимых процедур

- не являются языками общего назначения
- имеют низкую выразительность
- имеют сомнительную производительность
- ограничивают круг решаемых задач (но не жестко)

Экосистема

Экосистема

- достаточно бедна (не и с чего выбрать или нет инструментов вообще)
  - статический анализ кода
  - профилирование
  - дебаг
  - тестирование (моки, стабы, покрытие кода)
  - линтеры и фораматтеры

Экосистема

- достаточно бедна (не и с чего выбрать или нет инструментов вообще)
  - статический анализ кода
  - профилирование
  - дебаг
  - тестирование (моки, стабы, покрытие кода)
  - линтеры и фораматтеры
- нет first class citizen поддержки со стороны IDE
  - требуется подключение к БД
  - поиск использований работает не всегда

Best practice по работе с хранимками

Где моя хранимка, чувак?

Код вне системы контроля версий - это

Где моя хранимка, чувак?

Код вне системы контроля версий - это

- отсутствие единого места, где можно ознакомиться с кодом приложения

Где моя хранимка, чувак?

Код вне системы контроля версий - это

- отсутствие единого места, где можно ознакомиться с кодом приложения
- отсутствие историчности изменений

Где моя хранимка, чувак?

Код вне системы контроля версий - это

- отсутствие единого места, где можно ознакомиться с кодом приложения
- отсутствие историчности изменений
- закомментированный и неактуальный код

Где моя хранимка, чувак?

Код вне системы контроля версий - это

- отсутствие единого места, где можно ознакомиться с кодом приложения
- отсутствие историчности изменений
- закомментированный и неактуальный код
- сложность поддержки не production инфраструктуры (невоспроизводимость)

Где моя хранимка, чувак?

Код вне системы контроля версий - это

- отсутствие единого места, где можно ознакомиться с кодом приложения
- отсутствие историчности изменений
- закомментированный и неактуальный код
- сложность поддержки не production инфраструктуры (невоспроизводимость)
- невозможность восстановления кодовой базы на точку во времени

Как и где хранить?

В системе контроля версий так, как этого требует ваш инструмент:

- https://github.com/sqitchers/sqitch
- https://github.com/yandex/pgmigrate
- https://github.com/liquibase/liquibase
- https://github.com/flyway/flyway
- …
- Самописный инструмент

Есть хранимка...

```
create or replace function user_create_new2(
  in userName text,
  in lastName text,
  in age smallint,
  out outUserId integer,
  out outAge smallint
) returns record language plpgsql as
$function$
declare
  user_full_name text;
  age smallint;
  user_record record;
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
          outUserId := user_record.id;
          if user_record.user_name != trim(lastName || ' ' || userName) then
                    update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
          end if;
          if user_record.age != age then
                    update users set age = age where id = user_record.id returning age into outAge;
          end if;
          return;
  end if;
  if outUserId is null then
          user_full_name := trim(lastName || ' ' || userName);
          insert into users (user_name, age) values (user_full_name, age)
          returning id, age into outUserId, outAge;
          return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
$function$;
```

Нарушение KISS

```
create or replace function user_create_new2(
  in userName text,
  in lastName text,
  in age smallint,
  out outUserId integer,
  out outAge smallint
) returns record language plpgsql as
$function$
declare
  user_full_name text;
  age smallint;
  user_record record;
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
          outUserId := user_record.id;
          if user_record.user_name != trim(lastName || ' ' || userName) then
                    update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
          end if;
          if user_record.age != age then
                    update users set age = age where id = user_record.id returning age into outAge;
          end if;
          return;
  end if;
  if outUserId is null then
          user_full_name := trim(lastName || ' ' || userName);
          insert into users (user_name, age) values (user_full_name, age)
          returning id, age into outUserId, outAge;
          return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
$function$;
```

Нарушение SRP

```
create or replace function user_create_new2
...
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
       outUserId := user_record.id;
       if user_record.user_name != trim(lastName || ' ' || userName) then
              update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
       end if;
       if user_record.age != age then
              update users set age = age where id = user_record.id returning age into outAge;
       end if;
       return;
  end if;
  if outUserId is null then
       user_full_name := trim(lastName || ' ' || userName);
       insert into users (user_name, age) values (user_full_name, age)
       returning id, age into outUserId, outAge;
       return;
  end if;
  ...
```

Нарушение DRY

```
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
       outUserId := user_record.id;
       if user_record.user_name != trim(lastName || ' ' || userName) then
              update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
       end if;
       if user_record.age != age then
              update users set age = age where id = user_record.id returning age into outAge;
       end if;
       return;
  end if;
  if outUserId is null then
       user_full_name := trim(lastName || ' ' || userName);
       insert into users (user_name, age) values (user_full_name, age)
       returning id, age into outUserId, outAge;
       return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
```

Смешение stateless и stateful

```
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
        outUserId := user_record.id;
        if user_record.user_name != trim(lastName || ' ' || userName) then
               update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
        end if;
        if user_record.age != age then
               update users set age = age where id = user_record.id returning age into outAge;
        end if;
        return;
  end if;
  if outUserId is null then
        user_full_name := trim(lastName || ' ' || userName);
        insert into users (user_name, age) values (user_full_name, age)
        returning id, age into outUserId, outAge;
        return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
```

Несколько точек выхода

```
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
        outUserId := user_record.id;
        if user_record.user_name != trim(lastName || ' ' || userName) then
               update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
        end if;
        if user_record.age != age then
               update users set age = age where id = user_record.id returning age into outAge;
        end if;
        return;
  end if;
  if outUserId is null then
        user_full_name := trim(lastName || ' ' || userName);
        insert into users (user_name, age) values (user_full_name, age)
        returning id, age into outUserId, outAge;
        return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
```

Обработка “непонятного” исключения

```
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
       outUserId := user_record.id;
       if user_record.user_name != trim(lastName || ' ' || userName) then
            update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
       end if;
       if user_record.age != age then
            update users set age = age where id = user_record.id returning age into outAge;
       end if;
       return;
  end if;
  if outUserId is null then
       user_full_name := trim(lastName || ' ' || userName);
       insert into users (user_name, age) values (user_full_name, age)
       returning id, age into outUserId, outAge;
       return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
```

Рекурсивный вызов

```
begin
  select * into user_record from users where user_name = trim(lastName || ' ' || userName);
  if user_record.user_name is not null then
       outUserId := user_record.id;
       if user_record.user_name != trim(lastName || ' ' || userName) then
              update users set user_name = trim(lastName || ' ' || userName) where id = user_record.id;
       end if;
       if user_record.age != age then
              update users set age = age where id = user_record.id returning age into outAge;
       end if;
       return;
  end if;
  if outUserId is null then
       user_full_name := trim(lastName || ' ' || userName);
       insert into users (user_name, age) values (user_full_name, age)
       returning id, age into outUserId, outAge;
       return;
  end if;
exception when unique_violation then
  select f.outUserId, f.outAge into outUserId, outAge from user_create_new2(userName, lastName, age) f;
  return;
end;
```

Нам нужна магия рефакторинга...

Хранимка превращается...

```
create or replace function user_set(
	in i_first_name text,
	in i_last_name text,
	in i_age smallint,
	out o_id integer,
	out o_age smallint
) returns record language plpgsql as
$function$
begin
	insert into users as u (user_name, age)
	values (format_user_name(i_last_name, i_first_name), i_age)
	on conflict (user_name) do
	update set
		user_name = excluded.user_name,
		age = excluded.age
	returning id, age into o_id, o_age;
	return;
end;
$function$;
```

Хранимка превращается...

```
create or replace function user_set(
       in i_first_name text,
       in i_last_name text,
       in i_age smallint,
       out o_id integer,
       out o_age smallint
) returns record language plpgsql as
$function$
begin
       insert into users as u (user_name, age)
       values (format_user_name(i_last_name, i_first_name), i_age)
       on conflict (user_name) do
       update set
              user_name = excluded.user_name,
              age = excluded.age
       returning id, age into o_id, o_age;
       return;
end;
$function$;
```

А мы ничего не потеряли? Что говорят тесты?

Модульные тесты

Модульные тесты

- Для функций без состояния просто берем и тестируем (pgtap)

Модульные тесты

- Для функций без состояния просто берем и тестируем (pgtap)
- Для функций с состоянием:
  - Готовим фикстуры (глобальные или локальные) и тестируем (пишем сами)
  - Делаем моки и тестируем (pgmock + pgtap)

Покрытие и отчеты

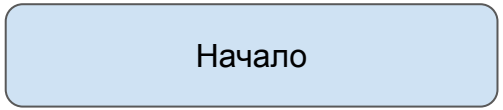
- Считаем покрытие? (костыли вокруг plpgsql_check, piggly или plprofiler)
- Готовим тестовые отчеты (tap-junit)

Avito

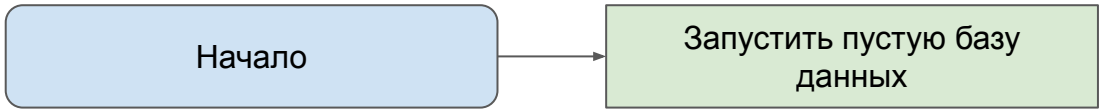
Интеграционные тесты

- Тестируем слой приложения по работе с базой данных поверх “настоящей” базы данных, например, в docker-контейнере
- Используем готовый инструментарий

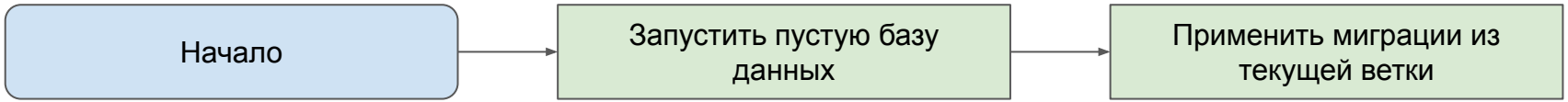
Автоматизация тестирования



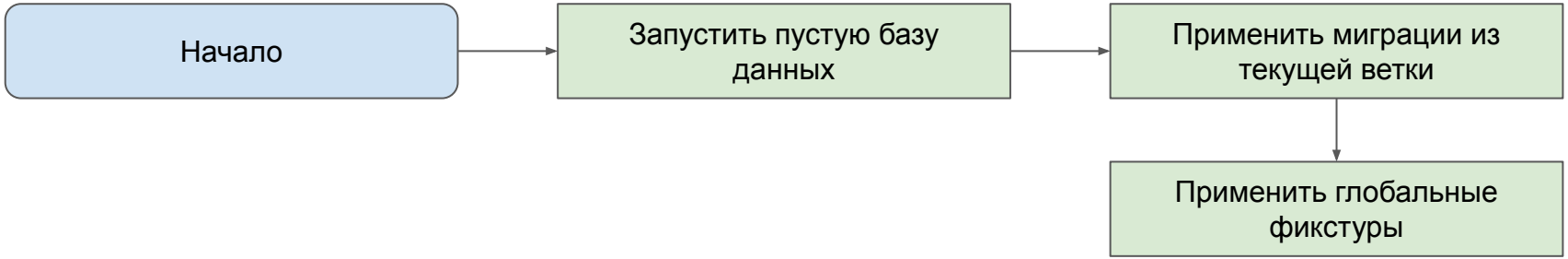
Автоматизация тестирования



Автоматизация тестирования

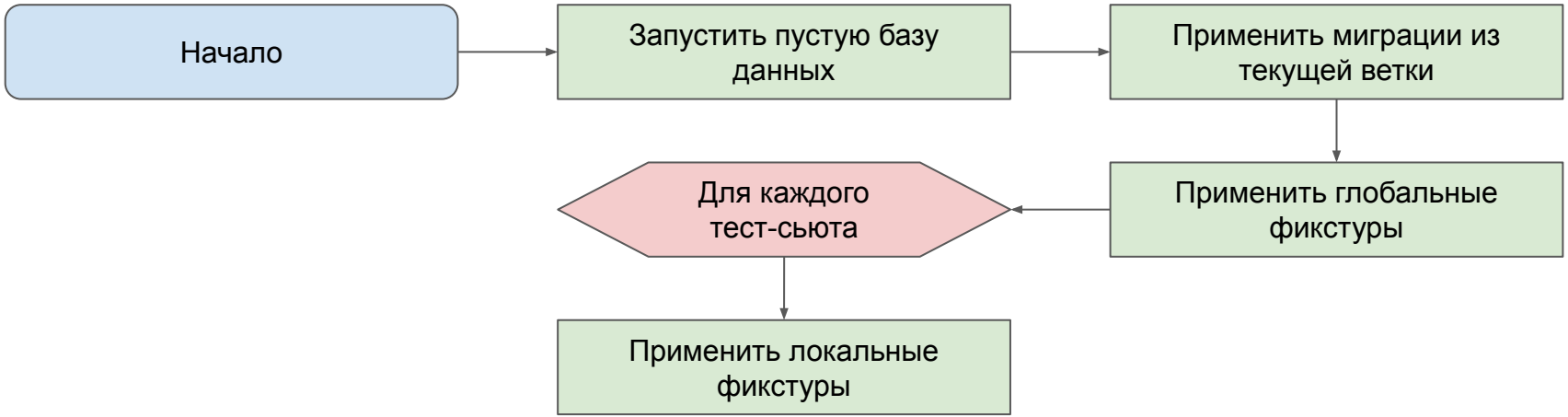


Автоматизация тестирования

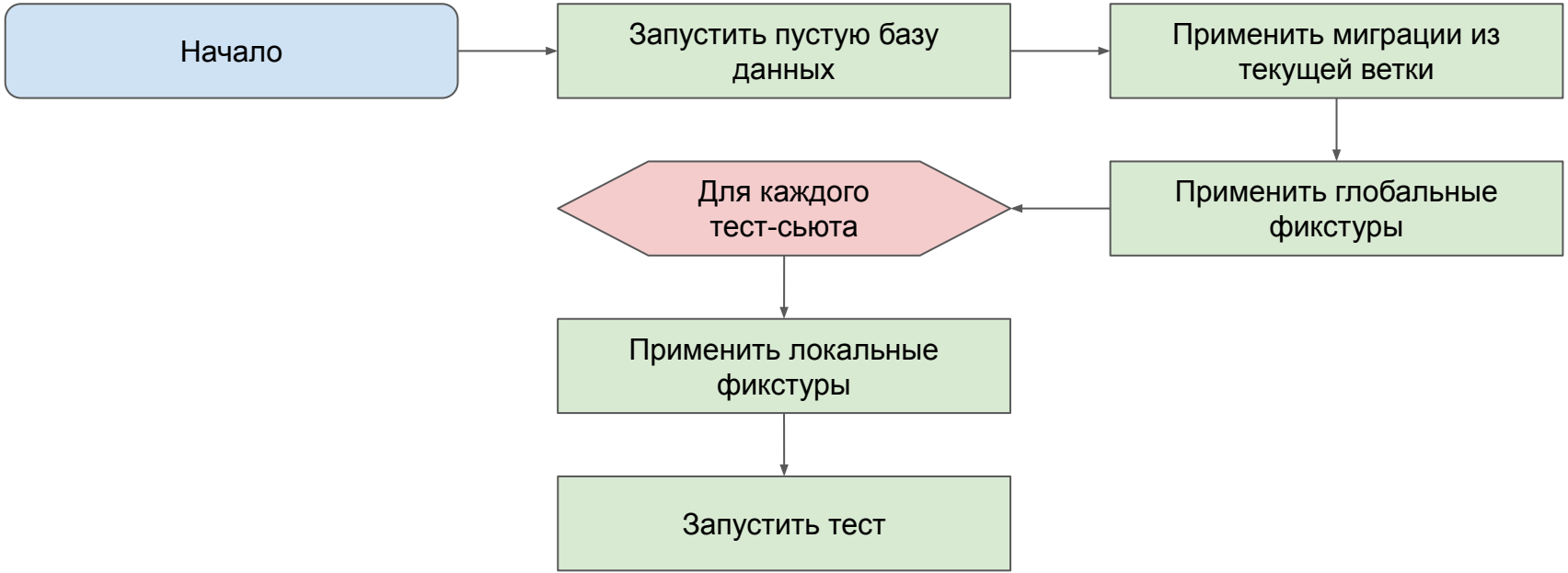


Автоматизация тестирования

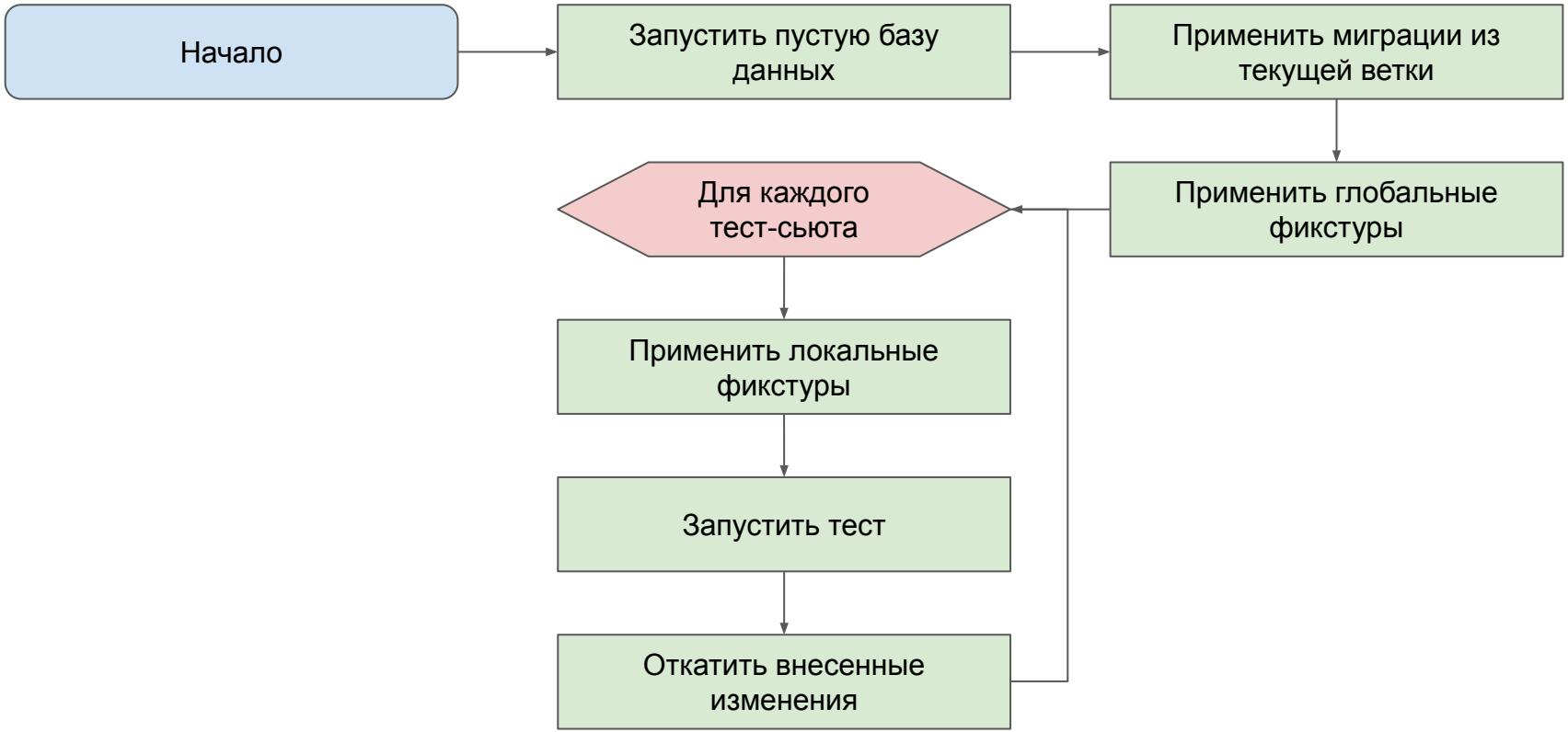
Автоматизация тестирования



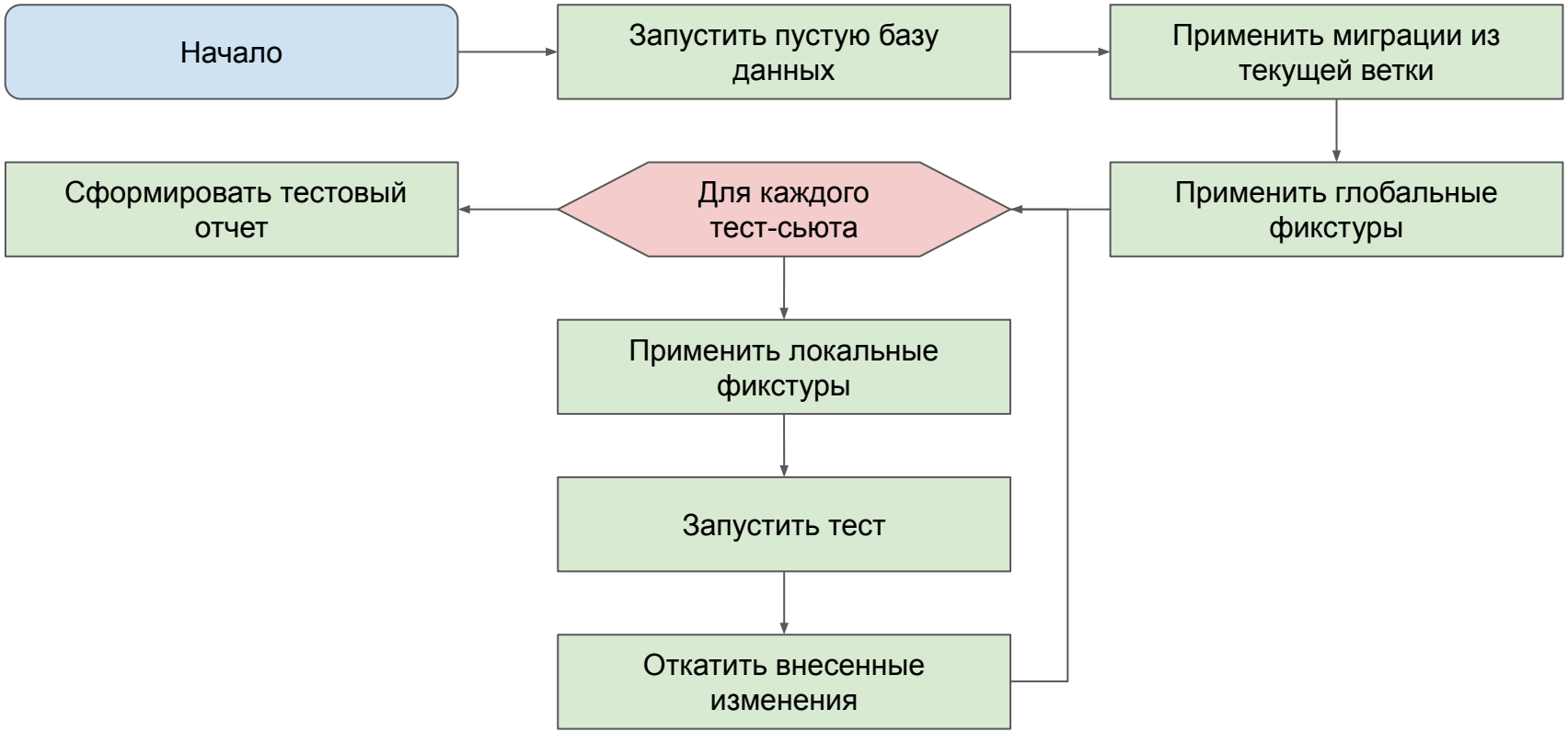
Автоматизация тестирования



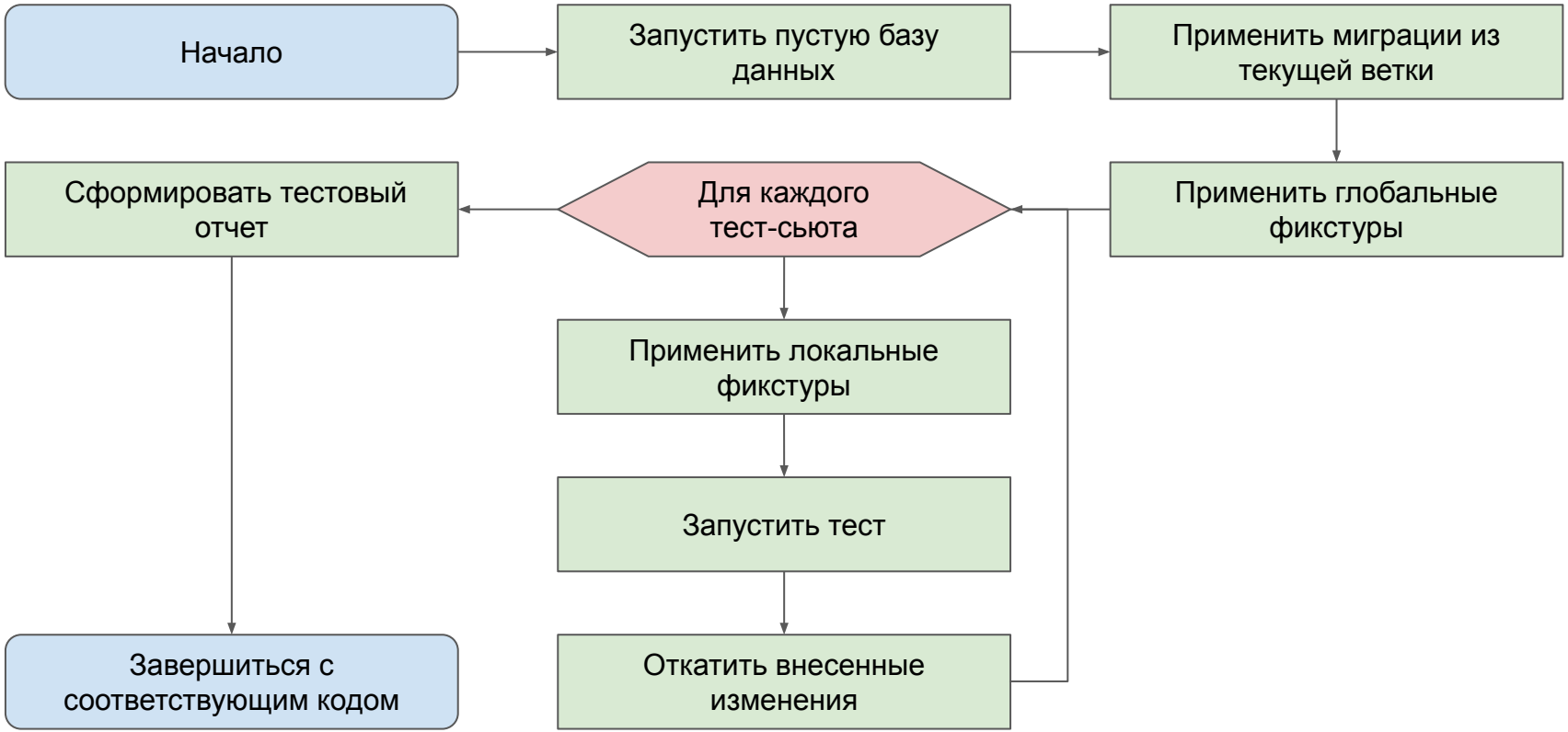
Автоматизация тестирования



Автоматизация тестирования

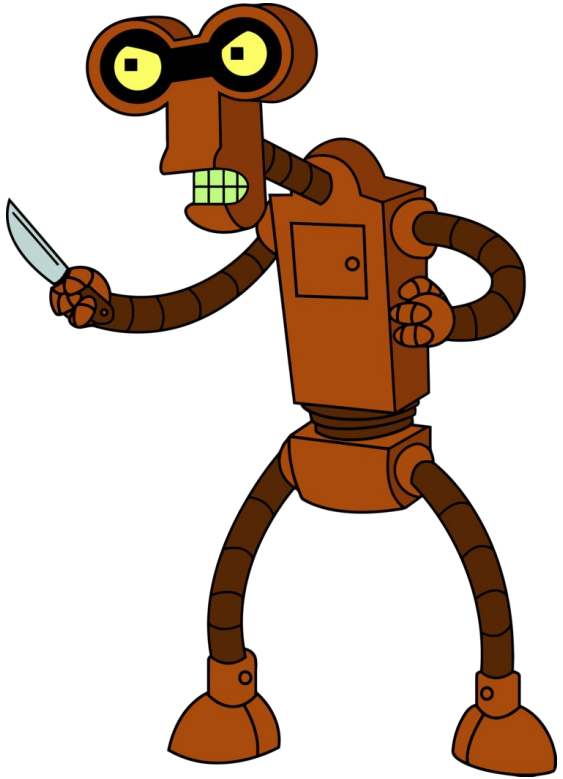


Автоматизация тестирования

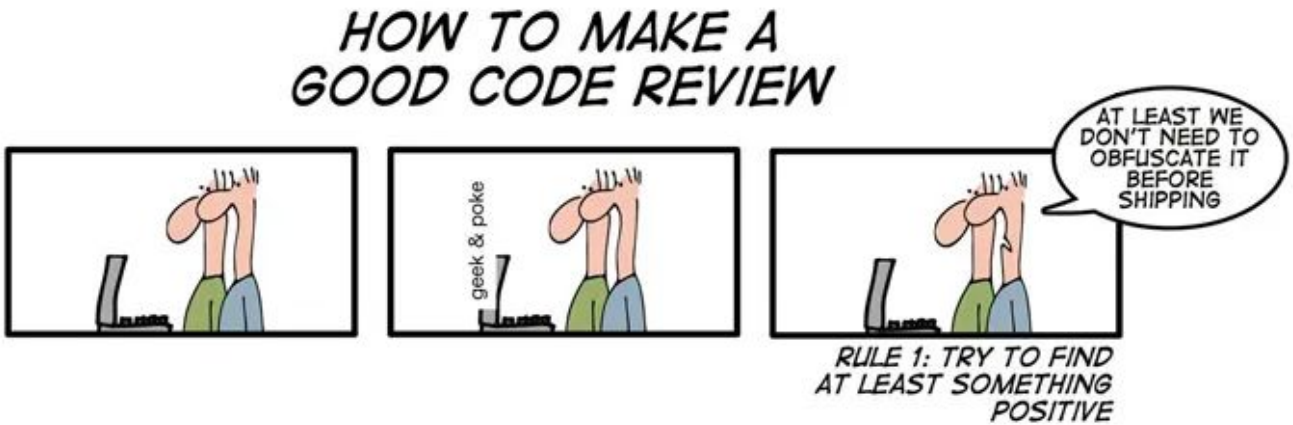


Автоматизация тестирования

Тесты обязательно должны запускаться на CI pipeline на каждый пул-реквест!



Code review


HOW TO MAKE A GOOD CODE REVIEW
geek & poke
AT LEAST WE DON'T NEED TO OBFUSCATE IT BEFORE SHIPPING
RULE 1: TRY TO FIND AT LEAST SOMETHING POSITIVE

Code review

- Обязателен

Code review

- Обязателен
- Хорошо бы иметь CSC и настроенные formatter-ы (pgFormatter, встроенные в IDE и др.) и linter-ы (???)

Code review

- Обязателен
- Хорошо бы иметь CSC и настроенные formatter-ы (pgFormatter, встроенные в IDE и др.) и linter-ы (???)
- Только после того, как автоматика сделала свое дело

Code review

- Обязателен
- Хорошо бы иметь CSC и настроенные formatter-ы (pgFormatter, встроенные в IDE и др.) и linter-ы (???)
- Только после того, как автоматика сделала свое дело
- Проводится компетентным специалистом

Code review

- Обязателен
- Хорошо бы иметь CSC и настроенные formatter-ы (pgFormatter, встроенные в IDE и др.) и linter-ы (???)
- Только после того, как автоматика сделала свое дело
- Проводится компетентным специалистом
- Блокирует слияние изменений в master (release) ветку

Production deploy тогда

Production deploy тогда

- Участие человека (DBA)

Production deploy тогда

- Участие человека (DBA)
- Обратнонесовместимые или несогласованные с кодом приложения изменения

Production deploy тогда

- Участие человека (DBA)
- Обратнонесовместимые или несогласованные с кодом приложения изменения
- Большое количество изменений (гигантские скрипты миграций)

Production deploy тогда

- Участие человека (DBA)
- Обратнонесовместимые или несогласованные с кодом приложения изменения
- Большое количество изменений (гигантские скрипты миграций)
- Применение изменений сразу “на всех”

Production deploy тогда

- Участие человека (DBA)
- Обратнонесовместимые или несогласованные с кодом приложения изменения
- Большое количество изменений (гигантские скрипты миграций)
- Применение изменений сразу “на всех”
- Downtime на время применения изменений

Production deploy тогда

- Участие человека (DBA)
- Обратнонесовместимые или несогласованные с кодом приложения изменения
- Большое количество изменений (гигантские скрипты миграций)
- Применение изменений сразу “на всех”
- Downtime на время применения изменений
- Отсутствие “скриптов отката”

Production deploy тогда

Production deploy сейчас

Production deploy сейчас

- Код до production доставлятся с помощью CI/CD pipeline

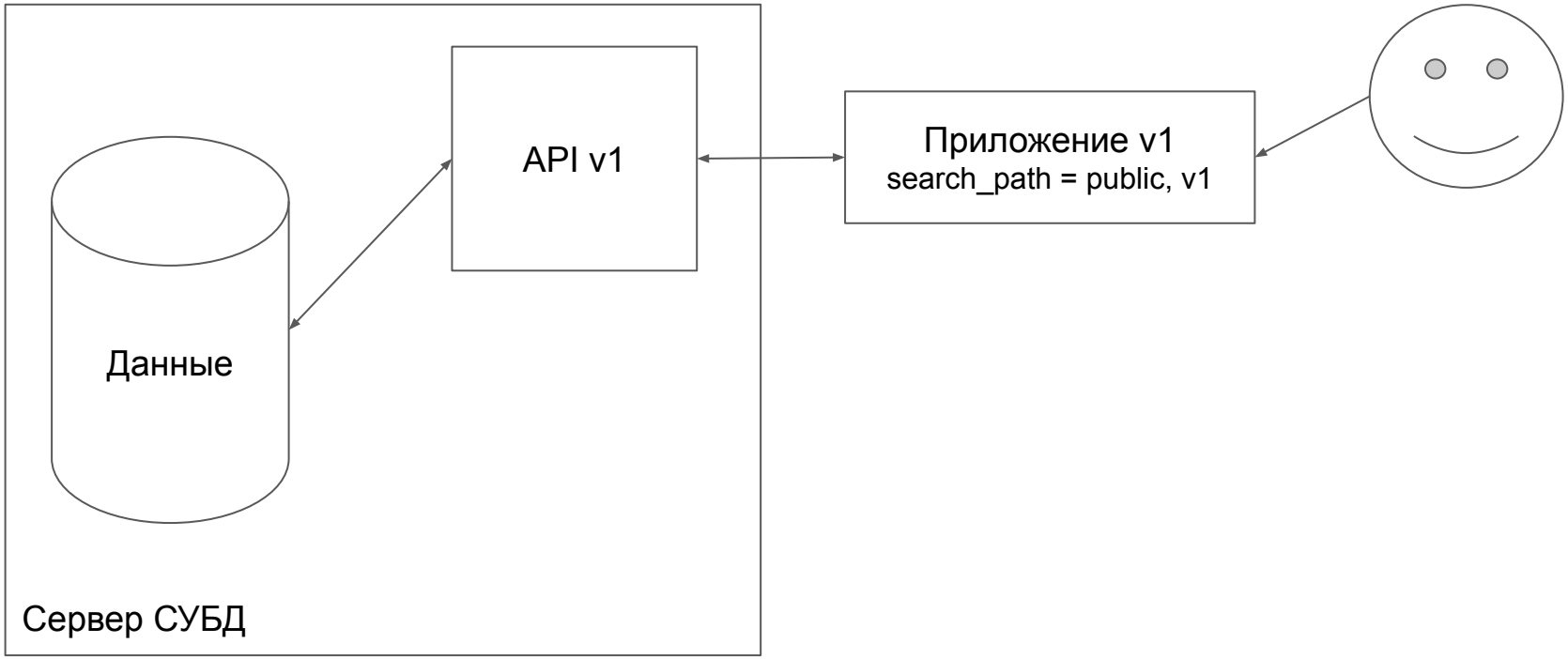
Production deploy сейчас

- Код до production доставлятся с помощью CI/CD pipeline
- Изменения выкладываются в production по готовности

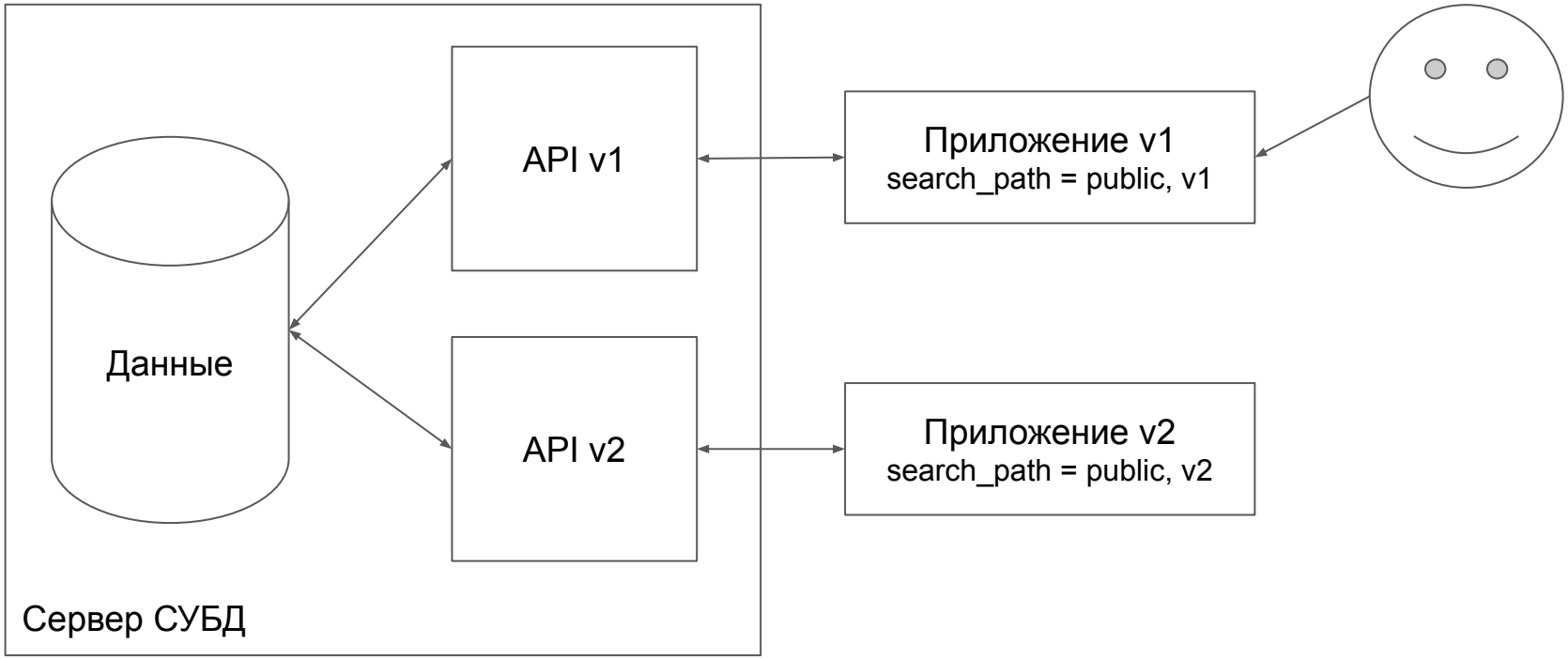
Production deploy сейчас

- Код до production доставлятся с помощью CI/CD pipeline
- Изменения выкладываются в production по готовности
- Код хранимых процедур версионируется

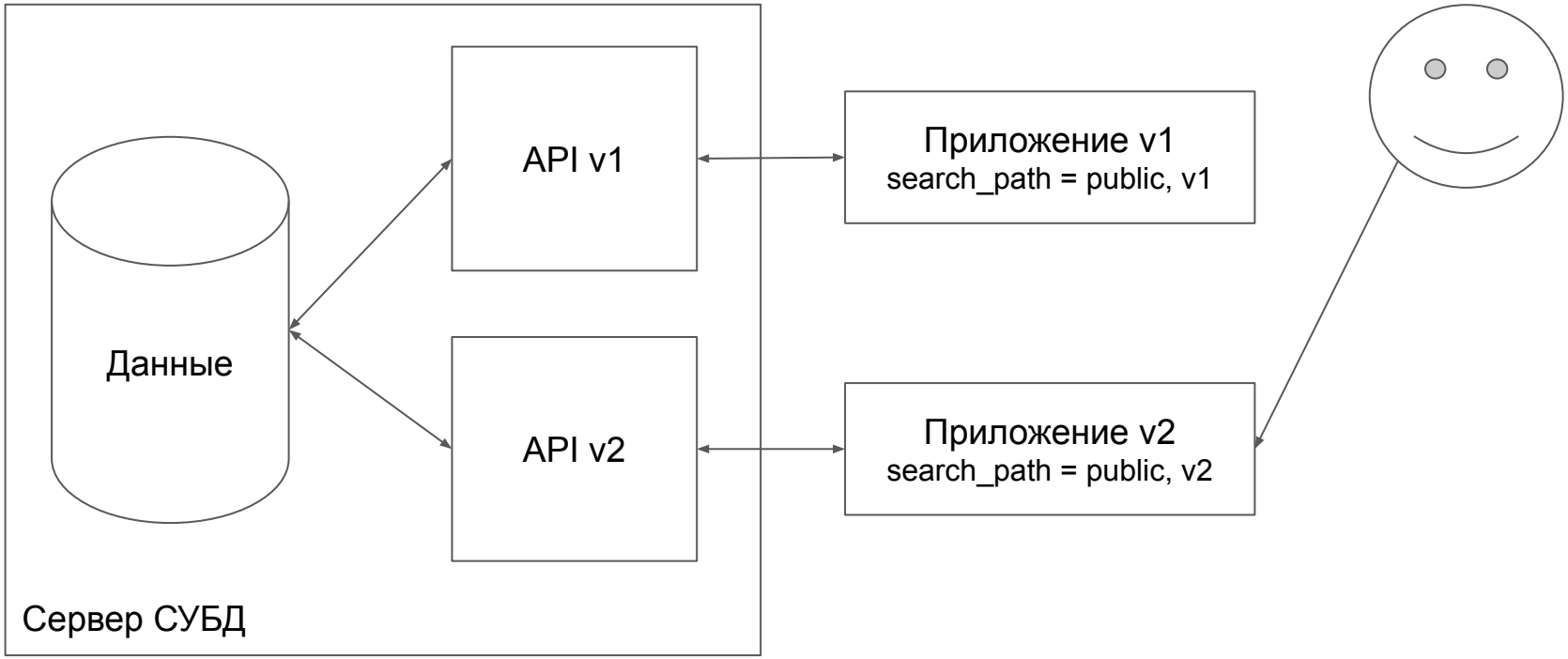
Версионируемый деплой



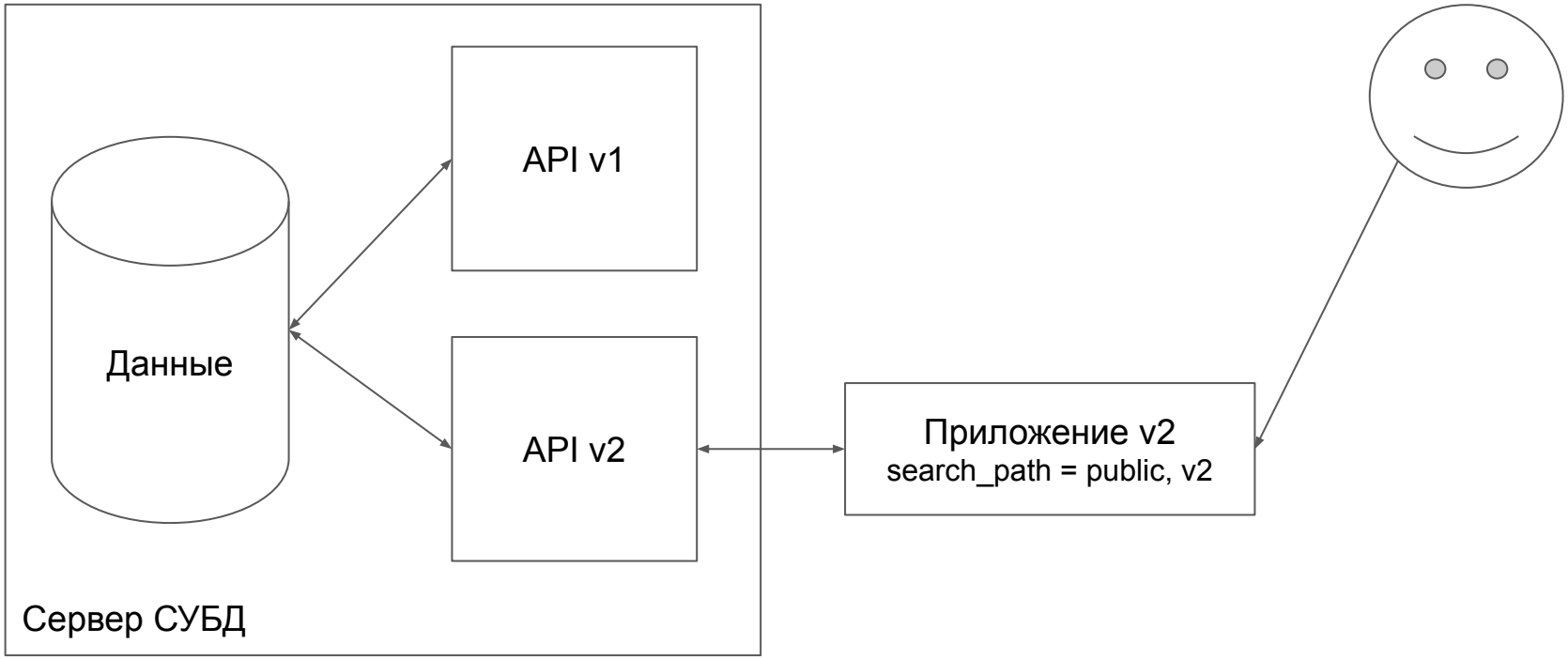
Версионируемый деплой



Версионируемый деплой



Версионируемый деплой



Production deploy сейчас

- Код до production доставлятся с помощью CI/CD pipeline
- Изменения выкладываются в production по готовности
- Код хранимых процедур версионируется
  - Механизм версионирования дает возможность делать canary deploy
  - Механизм версионирования дает возможность проводить обновления без downtime
  - Откат происходит мгновенно путем переключения на предыдущую версию кода

Production deploy сейчас

Итог

- Хранимки - инструмент:
  - специализированный
  - низкоуровневый
  - достаточно бедный (как язык и экосистема в целом)

Итог

- Хранимки - инструмент:
  - специализированный
  - низкоуровневый
  - достаточно бедный (как язык и экосистема в целом)
- Хранимки - код, поэтому:
  - должны хранится в системе контроля версий
  - к ним применимы все базовые принципы разработки ПО
  - поэтому их нужно тестировать
  - поэтому их нужно рефакторить
  - поэтому нужно проводить code review

Итог

- Хранимки - инструмент:
  - специализированный
  - низкоуровневый
  - достаточно бедный (как язык и экосистема в целом)
- Хранимки - код, поэтому:
  - должны хранится в системе контроля версий
  - к ним применимы все базовые принципы разработки ПО
  - поэтому их нужно тестировать
  - поэтому их нужно рефакторить
  - поэтому нужно проводить code review
- Последний этап активного участия человека - code review. Остальное должна делать автоматика

Ссылки

- Тестирование
  - https://github.com/theory/pgtap
  - https://github.com/avito-tech/pgmock
- Статический анализ и покрытие
  - https://github.com/okbob/plpgsql_check
  - https://github.com/kputnam/piggly
  - https://www.npmjs.com/package/tap-junit
- Форматтеры
  - https://github.com/darold/pgFormatter
- Версионирование и деплой
  - https://habr.com/ru/company/avito/blog/342946/

Avito

Спасибо за внимание!