DocOps: Строим шоссе к актуальной документации

Constructing up-to-date doc highway

Николай Поташников

Руководитель проектов, IT-архитектор
КУРС-ИТ

consulting@yandex.ru @nmpotashnikov

DevOops 2022

Популярные инструменты автоматического документирования

- JavaDoc, foodoc, bardoc…
- Отчёты о тестировании (Allure, …)
- Автоматическую документацию по спецификации API (GraphQL, OpenAPI, SOAP, …)

Какой процент документации можно ими автоматизировать?

Можно ли использовать аналогичные подходы для всей документации?

Можно и нужно!

В докладе я расскажу о практических приёмах непрерывного документирования, которые использую в работе.

Структура доклада

- Какая документация должна быть актуальна
- Инженерные практики, обеспечивающие актуальность документации
- Технические аспекты доставки документации

Насколько важна актуальность? Пример документирования карточки

Регистрационные данные

Логин

E-mail

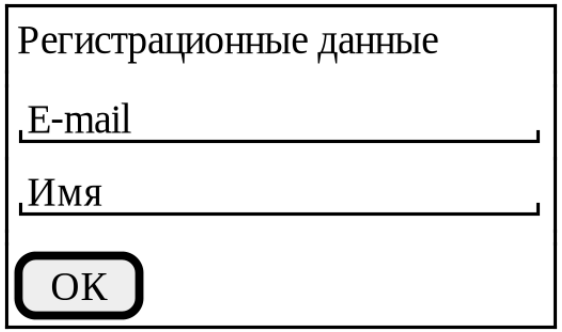
Имя для обращения

OK

Выдержка из документации

- `Пользовательское имя` – имя для входа в систему
- `E-mail` – адрес электронной почты
- `Полное имя` – имя для обращения

Прошло время



Документацию поменять забыли

- `Пользовательское имя` – имя для входа в систему
- `E-mail` – адрес электронной почты
- `Полное имя` – имя для обращения

А если аналогичное несоответствие есть в коде?

Сработает ли такой код на Typescript?

```
const login = 'ivanov1981'
const email = 'ivanov1981@bar.foo'
const fullName = 'Ivan Ivanov'

/* @ts-ignore */
outputUserInfo(email, fullName)

function outputUserInfo(login: string, email: string, fullName: string) {
  console.log(`login: ${login}\nemail: ${email}\nfull name: ${fullName}`)
}
```

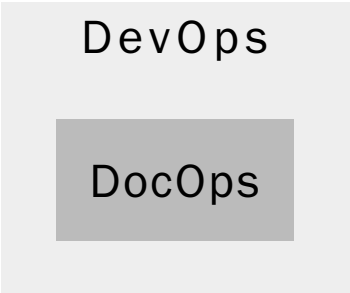
```
login: ivanov1981
email: Ivan Ivanov
full name: undefined
```

- Если пользователь ожидает от документации актуальности, значит она должна быть актуальной
- Требования к качеству кода и документации не должны отличаться

DocOps: варим актуальную документацию

WRITE THE DOCS

“DocOps” is like DevOps. Instead of applying broadly to software development, though, DocOps specifically applies to the creation, management, and release of documentation.



Документирование, разработка и доведение ИТ-продукта до конечного пользователя — это единый взаимоувязанный процесс.

Как обеспечить актуальность

- Автогенерировать готовые куски документации
- Тестировать актуальность

Источники формирования документации

- Код реализации, код тестов
- DSL, переиспользуемые в коде и в документации

  DSL — domain specific language — предметно-ориентированный язык программирования. Подробнее — Markus Voelter, Heisenbug, Питер 2021
- Результат работы приложения в тестовой или продуктивной среде

Объединим два предыдущих слайда в матрицу

	Автогенерация	**Тестирование актуальности**
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Какие-то клетки можно заполнить сразу

	Автогенерация	Тестирование актуальности
Код реализации, код тестов	Javadoc, foodoc, bardoc…	
DSL	SOAP, OpenAPI	
Результат работы приложения, в тестовой или продуктивной среде	Allure, отчёты по тестированию	

Для понимания дальнейших примеров требуется Asciidoc

```
Это *параграф*  ❶

.Исходный код
[source, java, indent=0]
----
include::some-file.java[tag=some-region] ❷
----

ifdef::is-administrator[Параграф, значимый только для администратора] ❸

ifndef::is-administrator[Параграф, лишний для администратора] ❸
```

❶ Задание параграфа, в котором слово «параграф» выделно жирным

❷ Вставка куска кода из java-файла

❸ Профилирование текста (отображение текста в зависимости от параметров сборки)

Результат предыдущего примера

Это **параграф**

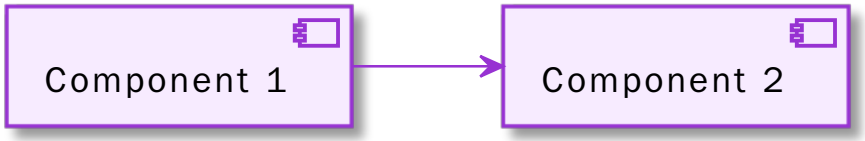
Исходный код

```
public static void main(String[] args) {
    System.out.println("Hello World!");
}
```

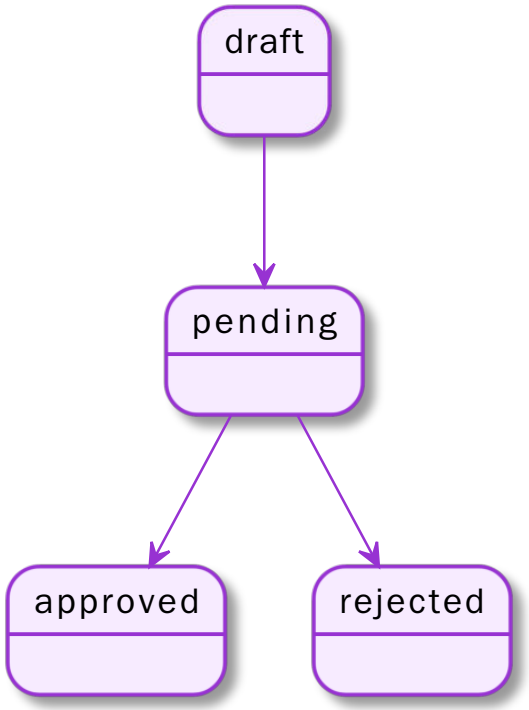
Параграф, значимый только для администратора

Диаграммы внутри Asciidoc

```
[plantuml, example, svg]
....
component "Component 1" as  component1
component "Component 2" as  component2
component1 --> component2
....
```



Рассмотрим кейс — Схема обработки счёта



Вариант 1. Автогенерация документации на основе кода

	Автогенерация	Тестирование актуальности
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Иногда достаточно просто поместить код в документацию

Ожидаемый вид документации

Далее приведен код реализации рабочего процесса работы со счётом

```
switch (state) {
  case 'draft':
    return ['pending'];
  case 'pending':
    return ['approved', 'rejected'];
}
return [];
```

Реализация

bill-processing.groovy

```
//tag::bill-process-implementation[] ❶
switch (state) {
  case 'draft':
    return ['pending'];
  case 'pending':
    return ['approved', 'rejected'];
}
return [];
...
//end::bill-process-implementation[] ❶
```

❶ Тэгирование нужной части кода

Код Asciidoc

```
include::bill-processing.groovy[tag=bill-process-implementation, indent=0]
```

А если попробовать красиво параметризовать тест?

```
private static Stream<String> provideCorrectTransitions() {
  def doc = """tag::correct-transitions[]
    [draft] --> [pending]
    [pending] --> [approved]
    [pending] --> [rejected]
    end::correct-transitions[]"""
  return Stream.of(doc.split('\n')[1..-2] as String[]);
}
```

Тогда в документации можем получить картинку

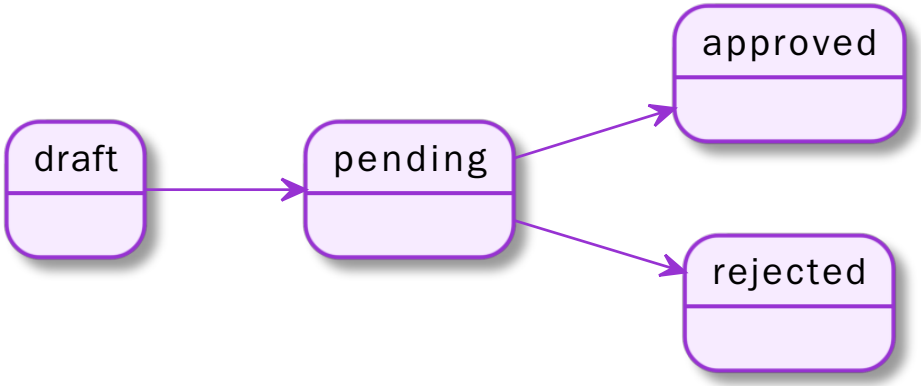


Схема рабочего процесса работы со счётом

Код Asciidoc

```
[plantuml,from-test,svg,width=50%]
....
left to right direction
state draft
include::../../demo/20-bill-server/bill-processing.groovy[tag=correct-transitions]
....
```

Другой пример генерации документации по коду: Jedis-Mock

Jedis-Mock позволяет мокировать вызовы Redis для целей тестирования

Задача

Отобразить все операции Redis и выделить реализованные

Решение

- Каждая реализованная операция аннотируется
- При генерации документации автоматически сравнивается общий перечень команд, возвращаемых Redis, и список аннотированных команд
- Для аннотированных и неаннотированных команд реализуется различное оформление

Результат

master jedis-mock / supported_operations.md Go to file ···

inponomarev Update list of supported operations Latest commit c16c531 on Jan 30 History

6 contributors

271 lines (241 sloc) | 5.14 KB Raw Blame

Supported operations:

Connection

✅ auth
✅ client
❌ echo
✅ hello
✅ ping
✅ quit
❌ reset
✅ select

Cluster

❌ cluster
❌ readonly
❌ readwrite

Вариант 2. Тестирование изменения кода

	Автогенерация	Тестирование актуальности
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Требования к алгоритму тестирования изменения кода

- Код, меняющий поведение программы, должен быть выделен
- Снимок (снэпшот) выделенного кода должен сохраняться в репозитории
- Документация должна содержать отметки, которые не позволяют сделать сборку, если соответствующий ей код изменился

Тэггируем нужные куски кода

```
// tag::correct-transition-provider[]
private static Stream<String> provideCorrectTransitions() {
  def doc = """ tag::correct-transitions[]
    draft --> pending
    pending --> approved
    pending --> rejected
    end::correct-transitions[] """
  return Stream.of(doc.split('\n')[1..-2] as String[]);
}
// end::correct-transition-provider[]
// tag::processing-follows-model[]
@ParameterizedTest
@MethodSource("provideCorrectTransitions")
void ProcessingFollowsModel(String correctTransition) {
  ...
}
// end::processing-follows-model[]
```

Объединяем тэггированные куски, bill-process-test.adoc

```
include::../bill-processing.groovy[tag=correct-transition-provider]
include::../bill-processing.groovy[tag=processing-follows-model]
```

Сравнение кода со снэпшотом (Approval Tests)

TestActual.checkSrcPartials.bill-process-test.received.txt - TestActual.checkSrcPartials.bill-process-test.approved.txt (/home/nmp/work/articles/docactual/DocOps-Heisenbug-2022-06...

Side-by-side viewer | Do not ignore | Highlight words | 1 difference

TestActual.checkSrcPartials.bill-process-test.received.txt (/home/nmp/work/articles/docactual/Do...

```
bill-process-test.adoc


 private static Stream&lt;String&gt; provideCorrectTransition
    draft --&gt; pending
    pending --&gt; approved
    pending --&gt; someNewState
    pending --&gt; rejected
  return Stream.of(doc.split('\n')[1..-2] as String[]);
}

@ParameterizedTest
@MethodSource("provideCorrectTransitions")
void ProcessingFollowsModel(String correctTransition) {
  def transition = correctTransition.split('--&gt;').collect
  billProcessor.setState(transition[0])
  assertThat(Arrays.asList(billProcessor.getPossibleStates() a
      , hasItems(transition[1]))
} .length = 0
```

TestActual.checkSrcPartials.bill-process-test.approved.txt (/home/nmp/work/articles/docactual/D...

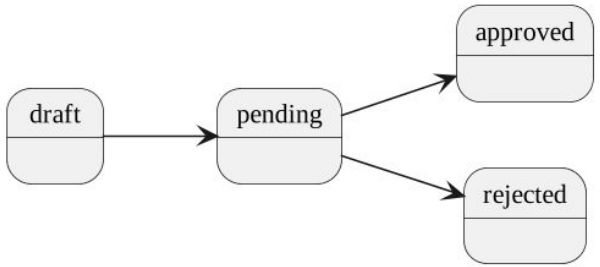
```
bill-process-test.adoc


 private static Stream&lt;String&gt; provideCorrectTransitions(
    draft --&gt; pending
    pending --&gt; approved
    pending --&gt; rejected
  return Stream.of(doc.split('\n')[1..-2] as String[]);
}

@ParameterizedTest
@MethodSource("provideCorrectTransitions")
void ProcessingFollowsModel(String correctTransition) {
  def transition = correctTransition.split('--&gt;').collect {
  billProcessor.setState(transition[0])
  assertThat(Arrays.asList(billProcessor.getPossibleStates() as
      , hasItems(transition[1]))
} .length = 0
```

```
:attribute-missing: warn
include::actual-test-adoc.adoc[]

== Bill processing workflow

ifndef::check-src-partials-bill-process-test[{check-src-partials-bill-process-test}]

Some text that we can't generate automatically.

[plantuml]
....
left to right direction
skinparam dpi 150
state draft
draft --> pending
pending --> approved
pending --> rejected
....
```

Bill processing workflow

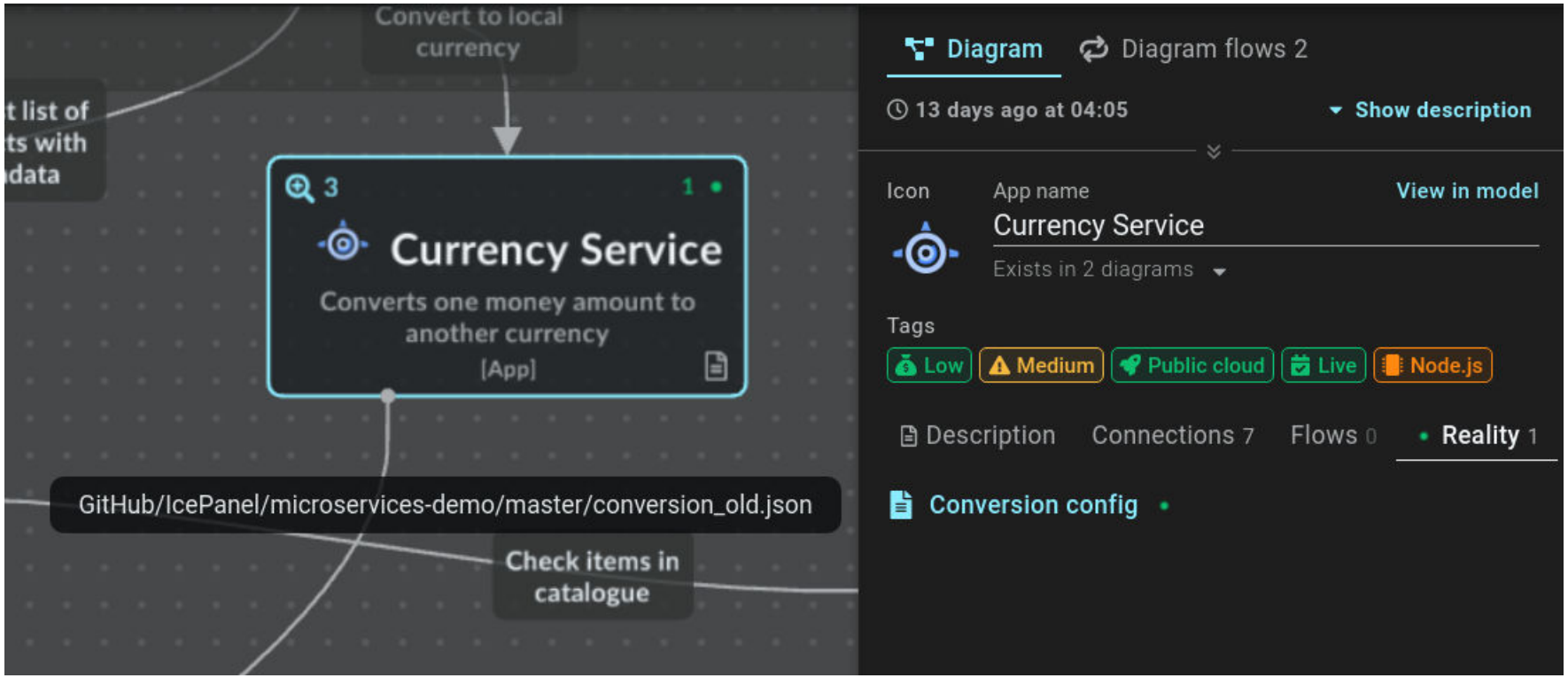
{check-src-partials-bill-process-test}

Some text that we can't generate automatically.



В каждом маркапе есть подход, которые без дополнительных расширений будет приводить к ошибке сборке при отсутствии требуемых атрибутов.

Связь с кодом в Icepanel



Вариант 3. Использование DSL для автогенерации документации

	Автогенерация	Тестирование актуальности
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Пример DSL для описания рабочего процесса работы со счётом

```
{
  "draft": ["pending"],
  "pending": [
    "rejected",
    "approved"
  ]
}
```

В коде

JSON-DSL → Map<String, Set<String>>

В документации

JSON-DSL → template → adoc, plantuml

Использование шаблонизаторов

Шаблон Liquid

```
{%- assign bl = "\n" -%}
{%-  for state in json -%}
    {%- for to_state in state[1] -%}
      {{ state[0] }} --> {{ to_state }}{{ bl }}
    {%- endfor -%}
{%- endfor -%}
```

```
draft --> pending
pending --> approved
pending --> rejected
```

Вариант 3а. Генерация документации на основе констант i18n

Пример файла с константами

```
{
  "billStates": {
    "draft": "Draft",
    "pending": "Pending",
    "approved": "Approved",
    "rejected": "Rejected"
  }
}
```

Пример файла с константами, адаптированного под Asciidoc

```
:t-billStates-childs-count-4:
:t-billStates-draft: Draft
:t-billStates-pending: Pending
:t-billStates-approved: Approved
:t-billStates-rejected: Rejected
```

Примерный вид отчёта

Возможные состояния счёта:

- Draft;
- Pending;
- Approved;
- Rejected.

```
.Возможные состояния счёта:
* {t-billStates-draft};
* {t-billStates-pending};
* {t-billStates-approved};
* {t-billStates-rejected}.
```

Аналогичным образом можно ввести константы для наименования действий

Вариант 4. Тестирование DSL

	Автогенерация	Тестирование актуальности
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Проверяем, что количество вложенных констант не изменилось

```
ifndef::t-billStates-childs-count-4[{t-billStates-childs-count-4}]
```

Примеры DSL, удобных для генерации документации

- Константы интернационализации
- BPMN
- GraphQL, OpenAPI, SOAP
- application.yaml
- GitHub/GitLab/Jenkins workflows
- …

Вариант 5. Генерация документации по артефактам сборки. Скриншоты

	Автогенерация	Тестирование актуальности
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Используемые нами правила получения скриншотов

- Готовится тестовый набор(ы) данных
- Скриншот создается только для форм, которые могут быть вызваны напрямую через URL
- Скриншоты делаются только с использованием параметризуемого framework’а тестирования (в нашем случае – JUnit5 + Selenide)

Скриншот, созданный по тестовым данным

DocOps: constructing up-to-date documentation highway — Demo Bills application

DevOops 2022

Bills

Bill Date	State	Payee	Actions
2022-31-05	Draft	Company A	TO PENDING
2022-31-05	Pending	Company B	TO APPROVED TO REJECTED
2022-31-05	Approved	Company B	
2022-31-05	Rejected		

Rows per page: 10 1-4 of 4

Вариант 6. Тестирование артефактов сборки. Проверка DOM-модели

	Автогенерация	Тестирование актуальности
Код реализации, код тестов		
DSL		
Результат работы приложения, в тестовой или продуктивной среде		

Пример реализации

1. Получение DOM-модели (аналогично созданию скриншота)
2. Создание запроса по DOM-модели
   - XPath: `//span[@class='state-value']`
   - GPath: `doc.depthFirst().findAll { it.name() == 'span' && it.@class == 'state-value'}`
   - …
3. Использование Approval Tests для проверки результата:

```
[Draft, Pending, Approved, Rejected]
```

- Подходов для контроля актуальности документации — множество. Основная задача — выбрать наиболее эффективный
- Архитектура проекта прямо влияет на возможность автоматизированного контроля актуальности документации

Сборка и доставка документации

Документация на Operations

- Разнородность используемых компонент
- Железку в *Testcontainers* не загрузишь
- Часто требуется документация верхнего уровня, чтобы понять, как в целом устроен процесс доставки программного продукта
- Требования информационной безопасности часто ведут к непрозрачности инфраструктуры развертывания

Матрица применима: Пример — автоматическая генерация из артефактов

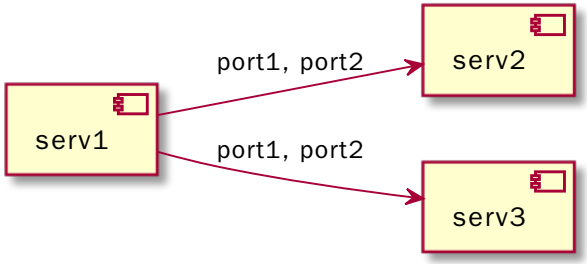
```
df -t ext4 --output=avail -B G | grep "[0-9]\+G$" | \
  sed 's/ //g' | \
  sed -e 's/\([0-9]\+G\)/!$available_hdd = "\1b"/g' \
  > to-include.pu
```

```
!$available_hdd = "15Gb"
```

```
package "This note" {
    component hdd [
        HDD
        \----
        Available: $available_hdd
    ]
}
```

DSL для документации. Описание сетевых взаимодействий с помощью Kotlin DSL

```kotlin
val app = "serv1" isServerWithIp "0.0.0.1" isFor "Сервер приложений"
val mail1 = "serv2" isServerWithIp "0.0.0.2" isFor "Почтовый сервер 1"
val mail2 = "serv3" isServerWithIp "0.0.0.3" isFor "Почтовый сервер 2"
setOf(app) linksTo setOf(mail1, mail2) isFor "Почтовые рассылки" via setOf("port1", "port2")
```



Инициатор	Компонента доступа	Порты	Примечание
serv1 (0.0.0.1)	serv2 (0.0.0.2), serv3 (0.0.0.3)	port1, port2	Почтовые рассылки
…			

Тестирование качества выходного документа

- Тестирование качества выходного документа — инструменты есть, например, Vale, но их возможности ограничены. Мы создавали собственный линтер
- Что-то протестировать нельзя — внешние ссылки
- Сложность smoke-тестирования: часть содержания не попала в выходной документ, вариант — выходной документ создался, но пустой. Если выходной формат docx или odt, то имеет смысл контролировать на соответствие схемы.

Что можно тестируем, в остальном нужна организация обратной связи (feedback)

Распределенный характер документации

Типовой подход — сборка документации из различных репозиториев. Предполагает наличие всего содержания документации в репозитории. А как быть с автогенерируемым содержанием, например, скриншотами?

Варианты

- Пересборка каждого репозитория при публикации
- Сборка документации из готовых артефактов
- Публикация документации в момент развертывания приложения. Требует наличия портала

Время сборки документации

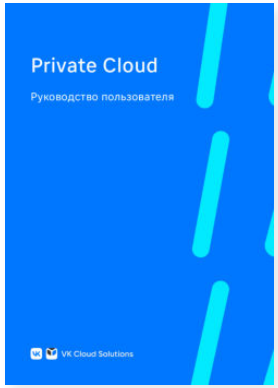
- *Заказчик* У нас долго собирается документация
- *Исполнитель* Ускорил в три раза
- *Заказчик* О! 8 часов — это классно

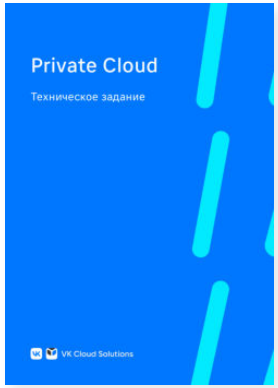
Требования к внешнему виду

Многие бросают DocOps, так как не могут получить документ в соответствим с требованиями заказчика.

Требования к документации — это стандартизация. Самая дешевая стандартизация — это автоматизация

Пример документации Private Cloud от VK





Private Cloud

Поиск

VK Cloud Solutions

Техническое задание

- **Общие положения**
- Требования к структуре и функционированию Системы
- Требования к способам и средствам связи для информационного обмена между компонентами системы
- Требования к режимам функционирования
- Требования к надёжности
- Подсистема потребителя облачных услуг
- Подсистема администрирования

Техническое задание

Программа и методика испытаний

Руководство администратора

Руководство пользователя

Техническое задание

Техническое задание / Общие положения

Общие положения

Полное наименование и условное обозначение

Полное наименование: Private Cloud.

Условное обозначение: Платформа.

Назначение и цели создания

Назначение

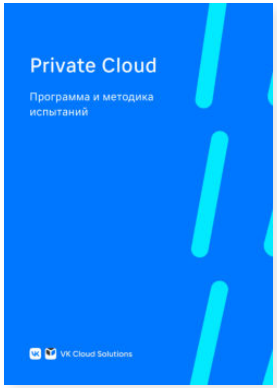
Private Cloud — это программный комплекс, предназначенный для построения частной облачной инфраструктуры с каталогом услуг, устанавливаемый на аппаратные мощности, предоставляемые Заказчиком в собственном в ЦОДе Заказчика (On premise).

Цели создания

Предоставление ресурсов для облачных вычислений внутренним пользователям Заказчика.

Содержание

Общие положения
Полное наименование и условное обозначение
Назначение и цели создания
Назначение
Цели создания
Определения, обозначения, сокращения



Как мы к этому шли

- Перевод всей документации на текстовый markup (в нашем случае — Asciidoctor)
- Перемещение исходников документации в репозиторий с кодом, объединение процессов CI/CD для приложения и документации
- Реализация механизмов автогенерации (с запуском в ручном режиме)
- Включение процессов автогенерации и тестирования актуальности в CI/CD
- Выработка стандартов DocOps

Все шаги алгоритма делались постепенно

Выводы

- Документация, от которой ожидают актуальности, должна быть актуальной
- Актуальная документация достижима на проекте любого масштаба, если он соответствует общепринятым требованиям по качеству и если команда готова работать открыто
- DocOps — начать получать пользу просто. В дальнейшнем, по мере взросления подхода, появляются новые задачи, но также и преимущества, которые дает DocOps для ИТ-проекта

Спасибо за внимание!