AVO

Интеграционное тестирование на Flutter и CI/CD

Ахмат Султанов

Mobile SDET
(iOS, Android, Cross-Platform)

Интеграционное тестирование на Flutter и CI/CD

Сегодня мы разберемся:

- Как использовать тестовый фреймворк для быстрого и стабильного выполнения интеграционных тестов.

Интеграционное тестирование на Flutter и CI/CD

Сегодня мы разберемся:

- Как использовать тестовый фреймворк для быстрого и стабильного выполнения интеграционных тестов.
- Динамическое создание и использование Android эмуляторов и iOS симуляторов на Mac mini.

Интеграционное тестирование на Flutter и CI/CD

Сегодня мы разберемся:

- Как использовать тестовый фреймворк для быстрого и стабильного выполнения интеграционных тестов.
- Динамическое создание и использование Android эмуляторов и iOS симуляторов на Mac mini.
- Подключение и настройка Android и Flutter Docker контейнров.

Быстрые и стабильные интеграционные тесты

- Как ускорить прогон тестового набора?

Быстрые и стабильные интеграционные тесты

- Как ускорить прогон тестового набора?
- Подготовка и очистка состояний приложения между тестами

Быстрые и стабильные интеграционные тесты

- Как ускорить прогон тестового набора?
- Подготовка и очистка состояний приложения между тестами
- Запуск тестов и автоматический перезапуск упавших тестов

Как ускорить прогон тестового набора?

```
import test_1_test.dart as test_1
import test_2_test.dart as test_2

void main() {
  test_1.main();
  test_2.main();
}
```

Запуск тестов из одного файла

Плюсы:
Больше не тратим время на сборку приложения для каждого тестового файла

Как ускорить прогон тестового набора?

```dart
import test_1_test.dart as test_1
import test_2_test.dart as test_2

void main() {
  test_1.main();
  test_2.main();
}
```

Запуск тестов из одного файла

Плюсы:
Больше не тратим время на сборку приложения для каждого тестового файла

Минусы:
Состояния между тестами не очищаются

Подготовка и очистка состояний приложения между тестами

Важный аспект в тестировании - обеспечение "предсказуемого" состояния приложения перед каждым тестом

Инициализация состояний приложения

```
class IntegrationTestHelper {
  Future<void> init() async {
    /* Вам нужно:
    - Зарегистрировать инстансы
    - Инициализировать Моки
    */
    final getIt = GetIt.instance;

    await getIt.allReady();

    // Пример замены инстансов на моковые
    getIt.unregister(instance: getIt<AppsFlyerService>());
    getIt.unregister(instance: getIt<OneSignalService>());

    getIt.registerSingleton<AppsFlyerService>(MockAppsFlyerService());
    getIt.registerSingleton<OneSignalService>(MockOneSignalService());
  }
}
```

Инициализация состояний приложения

```
class IntegrationTestHelper {
  Future<void> init() async {
    /* Вам нужно:
    - Зарегистрировать инстансы
    - Инициализировать Моки
    */
    final getIt = GetIt.instance;

    await getIt.allReady();

    // Пример замены инстансов на моковые
    getIt.unregister(instance: getIt<AppsFlyerService>());
    getIt.unregister(instance: getIt<OneSignalService>());

    getIt.registerSingleton<AppsFlyerService>(MockAppsFlyerService());
    getIt.registerSingleton<OneSignalService>(MockOneSignalService());
  }
}
```

Инициализация состояний приложения

```
class IntegrationTestHelper {
  Future<void> init() async {
    /* Вам нужно:
    - Зарегистрировать инстансы
    - Инициализировать Моки
    */
    final getIt = GetIt.instance;

    await getIt.allReady();

    // Пример замены инстансов на моковые
    getIt.unregister(instance: getIt<AppsFlyerService>());
    getIt.unregister(instance: getIt<OneSignalService>());

    getIt.registerSingleton<AppsFlyerService>(MockAppsFlyerService());
    getIt.registerSingleton<OneSignalService>(MockOneSignalService());
  }
}
```

Инициализация состояний приложения

```
class IntegrationTestHelper {
  Future<void> init() async {
    /* Вам нужно:
    - Зарегистрировать инстансы
    - Инициализировать Моки
    */
    final getIt = GetIt.instance;

    await getIt.allReady();

    // Пример замены инстансов на моковые
    getIt.unregister(instance: getIt<AppsFlyerService>());
    getIt.unregister(instance: getIt<OneSignalService>());

    getIt.registerSingleton<AppsFlyerService>(MockAppsFlyerService());
    getIt.registerSingleton<OneSignalService>(MockOneSignalService());
  }
}
```

Очистка состояний приложения

```
class IntegrationTestHelper {

  Future<void> dispose() async {
    /* Вам нужно:
    - Реализовать очищение состояний приложения
    */
  }
}
```

Очистка состояний приложения

```
extension IntegrationTesterExtention on WidgetTester {

  Future<void> performTestCleanUp(
    IntegrationTestHelper helper,
    IntegrationTestWidgetsFlutterBinding binding,
  ) async {
    await helper.dispose();

    binding.reset();
    binding.resetEpoch();
  }
}
```

Очистка состояний приложения

```
extension IntegrationTesterExtention on WidgetTester {

  Future<void> performTestCleanUp(
    IntegrationTestHelper helper,
    IntegrationTestWidgetsFlutterBinding binding,
  ) async {
    await helper.dispose();

    binding.reset();
    binding.resetEpoch();
  }
}
```

Запуск тестов

```
class IntegrationTestHelper {

Future<void> runTest({
    required Future<Null> Function() run,
    required Future<Null> Function() after,
  }) async {
    try {
      await run();
    } finally {
      await after();
    }
  }
}
```

Автоматический перезапуск упавших тестов

```
class IntegrationTestHelper {
  Future<void> runTest({
  required Future<void> Function() run,
  required Future<void> Function() after,
  int retries = 1,
  }) async {
    int currentAttempt = 0;
    dynamic lastError;

    bool hasError;

    do {
      try {
        await run();

        hasError = false;
        currentAttempt++;
      } catch (catched) {

        hasError = true;
        lastError = catched;
        currentAttempt++;
      } finally {
        await after();
      }
    } while (hasError && currentAttempt <= retries);

      if (hasError) {
       throw lastError.toString();
      }
    }
  }}
```

Автоматический перезапуск упавших тестов

```
class IntegrationTestHelper {
  Future<void> runTest({
  required Future<void> Function() run,
  required Future<void> Function() after,
  int retries = 1,
  }) async {
    int currentAttempt = 0;
    dynamic lastError;

    bool hasError;

    do {
      try {
        await run();

        hasError = false;
        currentAttempt++;
      } catch (catched) {

        hasError = true;
        lastError = catched;
        currentAttempt++;
      } finally {
        await after();
      }
    } while (hasError && currentAttempt <= retries);

      if (hasError) {
       throw lastError.toString();
      }
    }
  }}
```

Автоматический перезапуск упавших тестов

```
class IntegrationTestHelper {
  Future<void> runTest({
  required Future<void> Function() run,
  required Future<void> Function() after,
  int retries = 1,
  }) async {
    int currentAttempt = 0;
    dynamic lastError;

    bool hasError;

    do {
      try {
        await run();

        hasError = false;
        currentAttempt++;
      } catch (catched) {

        hasError = true;
        lastError = catched;
        currentAttempt++;
      } finally {
        await after();
      }
    } while (hasError && currentAttempt <= retries);

      if (hasError) {
       throw lastError.toString();
      }
    }
  }}
```

Автоматический перезапуск упавших тестов

```
class IntegrationTestHelper {
  Future<void> runTest({
  required Future<void> Function() run,
  required Future<void> Function() after,
  int retries = 1,
  }) async {
    int currentAttempt = 0;
    dynamic lastError;

    bool hasError;

    do {
      try {
        await run();

        hasError = false;
        currentAttempt++;
      } catch (catched) {

        hasError = true;
        lastError = catched;
        currentAttempt++;
      } finally {
        await after();
      }
    } while (hasError && currentAttempt <= retries);

      if (hasError) {
       throw lastError.toString();
      }
    }
  }}
```

Автоматический перезапуск упавших тестов

```
class IntegrationTestHelper {
  Future<void> runTest({
  required Future<void> Function() run,
  required Future<void> Function() after,
  int retries = 1,
  }) async {
    int currentAttempt = 0;
    dynamic lastError;

    bool hasError;

    do {
      try {
        await run();

        hasError = false;
        currentAttempt++;
      } catch (catched) {

        hasError = true;
        lastError = catched;
        currentAttempt++;
      } finally {
        await after();
      }
    } while (hasError && currentAttempt <= retries);

       if (hasError) {
        throw lastError.toString();
       }
    }
  }}
```

Автоматический перезапуск упавших тестов

```
class IntegrationTestHelper {
  Future<void> runTest({
  required Future<void> Function() run,
  required Future<void> Function() after,
  int retries = 1,
  }) async {
    int currentAttempt = 0;
    dynamic lastError;

    bool hasError;

    do {
      try {
        await run();

        hasError = false;
        currentAttempt++;
      } catch (catched) {

        hasError = true;
        lastError = catched;
        currentAttempt++;
      } finally {
        await after();
      }
    } while (hasError && currentAttempt <= retries);

      if (hasError) {
      throw lastError.toString();
      }
    }
  }}
```

Как будет выглядеть тест?

```
void main() {
  final binding = IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  binding.framePolicy = LiveTestWidgetsFlutterBindingFramePolicy.fullyLive;

  final IntegrationTestHelper helper = IntegrationTestHelper();

  testWidgets(('Test'), (tester) async {
    await helper.runTest(
      run: () async {
        await helper.init();

        // Ваша тестовая логика

    }, after: () async {
        await tester.performTestCleanUp(
          helper,
          binding,
        );
    });
  });
}
```

Как будет выглядеть тест?

```
void main() {
  final binding = IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  binding.framePolicy = LiveTestWidgetsFlutterBindingFramePolicy.fullyLive;

  final IntegrationTestHelper helper = IntegrationTestHelper();

  testWidgets(('Test'), (tester) async {
    await helper.runTest(
      run: () async {
        await helper.init();

        // Ваша тестовая логика

    }, after: () async {
        await tester.performTestCleanUp(
          helper,
          binding,
        );
    });
  });
}
```

Итоги первой части доклада, мы научились:

- Запускать тесты из одного файла, очищать состояния между тестами, устраняя необходимость пересборки приложения.

Итоги первой части доклада, мы научились:

- Запускать тесты из одного файла, очищать состояния между тестами, устраняя необходимость пересборки приложения.
- Перезапускать упавшие тесты, автоматически.

Динамическое создание и использование Android эмуляторов и iOS симуляторов на Mac mini

- Использование Android-эмуляторов на Mac mini.

Динамическое создание и использование Android эмуляторов и iOS симуляторов на Mac mini

- Использование Android-эмуляторов на Mac mini.
- Использование iOS-симуляторов на Mac mini.

Динамическое создание и использование Android эмуляторов и iOS симуляторов на Mac mini

- Использование Android-эмуляторов на Mac mini.
- Использование iOS-симуляторов на Mac mini.
- Запуск тестов с использованием нескольких девайсов.

Создание Android эмулятора

```
#!/bin/bash
CI_JOB_ID="$1"
FILE="$2"

flutter emulators --create --name ${CI_JOB_ID}

flutter emulators --launch ${CI_JOB_ID}

EMULATOR=$(adb devices | grep emulator | tail -n 1 | awk '{print $1}')
echo $EMULATOR > ${FILE}.txt

echo $EMULATOR
```

Создание Android эмулятора

```
#!/bin/bash
CI_JOB_ID="$1"
FILE="$2"

flutter emulators --create --name ${CI_JOB_ID}

flutter emulators --launch ${CI_JOB_ID}

EMULATOR=$(adb devices | grep emulator | tail -n 1 | awk '{print $1}')
echo $EMULATOR > ${FILE}.txt

echo $EMULATOR
```

Создание Android эмулятора

```
#!/bin/bash
CI_JOB_ID="$1"
FILE="$2"

flutter emulators --create --name ${CI_JOB_ID}

flutter emulators --launch ${CI_JOB_ID}

EMULATOR=$(adb devices | grep emulator | tail -n 1 | awk '{print $1}')
echo $EMULATOR > ${FILE}.txt

echo $EMULATOR
```

Ожидание полной загрузки эмулятора

```
#!/bin/bash
EMULATOR="$1"

echo "Использование эмулятора: ${EMULATOR}"
adb -s "$EMULATOR" wait-for-device
BOOT_COMPLETE=$(adb -s "$EMULATOR" shell getprop sys.boot_completed | tr -d '\r')

echo "Ожидание загрузки эмулятора (это может занять несколько минут)"
while [ "$BOOT_COMPLETE" != "1" ]; do
  sleep 5
  echo "$(date +%T) ждем загрузки эмулятора..."
  BOOT_COMPLETE=$(adb -s "$EMULATOR" shell getprop sys.boot_completed | tr -d '\r')
done

adb -s "$EMULATOR" shell input keyevent 82
echo 'Эмулятор полностью загружен и готов к использованию.'
```

Ожидание полной загрузки эмулятора

```
#!/bin/bash
EMULATOR="$1"

echo "Использование эмулятора: ${EMULATOR}"
adb -s "$EMULATOR" wait-for-device
BOOT_COMPLETE=$(adb -s "$EMULATOR" shell getprop sys.boot_completed | tr -d '\r')

echo "Ожидание загрузки эмулятора (это может занять несколько минут)"
while [ "$BOOT_COMPLETE" != "1" ]; do
  sleep 5
  echo "$(date +%T) ждем загрузки эмулятора..."
  BOOT_COMPLETE=$(adb -s "$EMULATOR" shell getprop sys.boot_completed | tr -d '\r')
done

adb -s "$EMULATOR" shell input keyevent 82
echo 'Эмулятор полностью загружен и готов к использованию.'
```

Ожидание полной загрузки эмулятора

```
#!/bin/bash
EMULATOR="$1"

echo "Использование эмулятора: ${EMULATOR}"
adb -s "$EMULATOR" wait-for-device
BOOT_COMPLETE=$(adb -s "$EMULATOR" shell getprop sys.boot_completed | tr -d '\r')

echo "Ожидание загрузки эмулятора (это может занять несколько минут)"
while [ "$BOOT_COMPLETE" != "1" ]; do
  sleep 5
  echo "$(date +%T) ждем загрузки эмулятора..."
  BOOT_COMPLETE=$(adb -s "$EMULATOR" shell getprop sys.boot_completed | tr -d '\r')
done

adb -s "$EMULATOR" shell input keyevent 82
echo 'Эмулятор полностью загружен и готов к использованию.'
```

Пример CI-скрипта

```
integration_test_android:
  before_script:
    - EMULATOR=$(bash ./create_emulator.sh ${CI_JOB_ID} “emulator”)
    - echo $EMULATOR
    - bash ./wait_for_emulator_boot.sh “$EMULATOR”
  script:
    - flutter test integration_test_folder/integration_test_file_name.dart -d ${EMULATOR}
  after_script:
    - EMULATOR=$(cat ./emulator.txt)
    - adb -s ${EMULATOR} emu kill
    - cd ~/.android/avd
    - rm -rf ${CI_JOB_ID}.*
```

Пример CI-скрипта

```
integration_test_android:
  before_script:
    - EMULATOR=$(bash ./create_emulator.sh ${CI_JOB_ID} “emulator”)
    - echo $EMULATOR
    - bash ./wait_for_emulator_boot.sh “$EMULATOR”
  script:
    - flutter test integration_test_folder/integration_test_file_name.dart -d ${EMULATOR}
  after_script:
    - EMULATOR=$(cat ./emulator.txt)
    - adb -s ${EMULATOR} emu kill
    - cd ~/.android/avd
    - rm -rf ${CI_JOB_ID}.*
```

Пример CI-скрипта

```
integration_test_android:
  before_script:
    - EMULATOR=$(bash ./create_emulator.sh ${CI_JOB_ID} “emulator”)
    - echo $EMULATOR
    - bash ./wait_for_emulator_boot.sh “$EMULATOR”
   script:
    - flutter test integration_test_folder/integration_test_file_name.dart -d ${EMULATOR}
  after_script:
    - EMULATOR=$(cat ./emulator.txt)
    - adb -s ${EMULATOR} emu kill
    - cd ~/.android/avd
    - rm -rf ${CI_JOB_ID}.*
```

Создание iOS симулятора

```
#!/bin/bash
MODEL="$1"
FILE="$2"

DEVICE_NAME="Test-iPhone-${MODEL}"
DEVICE_MODEL="com.apple.CoreSimulator.SimDeviceType.iPhone-${MODEL}"
DEVICE_OS="iOS16.4"


UUID=$(xcrun simctl create ${DEVICE_NAME} ${DEVICE_MODEL} ${DEVICE_OS})
echo $UUID > ${FILE}.txt

xcrun simctl bootstatus $UUID -b

echo $UUID
```

Создание iOS симулятора

```
#!/bin/bash
MODEL="$1"
FILE="$2"

DEVICE_NAME="Test-iPhone-${MODEL}"
DEVICE_MODEL="com.apple.CoreSimulator.SimDeviceType.iPhone-${MODEL}"
DEVICE_OS="iOS16.4"

UUID=$(xcrun simctl create ${DEVICE_NAME} ${DEVICE_MODEL} ${DEVICE_OS})
echo $UUID > ${FILE}.txt

xcrun simctl bootstatus $UUID -b

echo $UUID
```

Создание iOS симулятора

```
#!/bin/bash
MODEL="$1"
FILE="$2"

DEVICE_NAME="Test-iPhone-${MODEL}"
DEVICE_MODEL="com.apple.CoreSimulator.SimDeviceType.iPhone-${MODEL}"
DEVICE_OS="iOS16.4"


UUID=$(xcrun simctl create ${DEVICE_NAME} ${DEVICE_MODEL} ${DEVICE_OS})
echo $UUID > ${FILE}.txt

xcrun simctl bootstatus $UUID -b

echo $UUID
```

Создание iOS симулятора

```
#!/bin/bash
MODEL="$1"
FILE="$2"

DEVICE_NAME="Test-iPhone-${MODEL}"
DEVICE_MODEL="com.apple.CoreSimulator.SimDeviceType.iPhone-${MODEL}"
DEVICE_OS="iOS16.4"


UUID=$(xcrun simctl create ${DEVICE_NAME} ${DEVICE_MODEL} ${DEVICE_OS})
echo $UUID > ${FILE}.txt

xcrun simctl bootstatus $UUID -b

echo $UUID
```

Пример CI-скрипта

```
integration_test_ios:
  before_script:
    - cd ios
    - pod update
    - cd ../
    - UUID=$(bash ./create_simulator.sh "13-Pro-Max" "simulator")
    - echo $UUID
  script:
    - flutter test integration_test_folder/integration_test_file_name.dart -d $UUID
  after_script:
    - UUID=$(cat ./simulator.txt)
    - xcrun simctl shutdown $UUID
    - xcrun simctl delete $UUID
```

Пример CI-скрипта

```
integration_test_ios:
  before_script:
    - cd ios
    - pod update
    - cd ../
    - UUID=$(bash ./create_simulator.sh "13-Pro-Max" "simulator")
    - echo $UUID
  script:
    - flutter test integration_test_folder/integration_test_file_name.dart -d $UUID
  after_script:
    - UUID=$(cat ./simulator.txt)
    - xcrun simctl shutdown $UUID
    - xcrun simctl delete $UUID
```

Пример CI-скрипта

```
integration_test_ios:
  before_script:
    - cd ios
    - pod update
    - cd ../
    - UUID=$(bash ./create_simulator.sh "13-Pro-Max" "simulator")
    - echo $UUID
  script:
    - flutter test integration_test_folder/integration_test_file_name.dart -d $UUID
  after_script:
    - UUID=$(cat ./simulator.txt)
    - xcrun simctl shutdown $UUID
    - xcrun simctl delete $UUID
```

Что делать, если нужно запустить прогон сразу на нескольких девайсах?

Скрипт ожидания установки приложения на Android

```
function waitForAppInstallationAndroid {
  local DEVICE=$1
  local PACKAGE_NAME=$2

  while true; do
    adb -s $DEVICE shell pm list packages | grep "$PACKAGE_NAME" &> /dev/null

    if [ $? -eq 0 ]; then
      echo "Приложение установлено на $DEVICE"
      break
    else
      echo "Ожидание установки приложения на $DEVICE…"
      sleep 5
    fi
  done
}
```

Скрипт ожидания установки приложения на Android

```
function waitForAppInstallationAndroid {
  local DEVICE=$1
  local PACKAGE_NAME=$2

  while true; do
    adb -s $DEVICE shell pm list packages | grep "$PACKAGE_NAME" &> /dev/null

    if [ $? -eq 0 ]; then
      echo "Приложение установлено на $DEVICE"
      break
    else
      echo "Ожидание установки приложения на $DEVICE…"
      sleep 5
    fi
  done
}
```

Скрипт ожидания установки приложения на Android

```
function waitForAppInstallationAndroid {
  local DEVICE=$1
  local PACKAGE_NAME=$2

  while true; do
    adb -s $DEVICE shell pm list packages | grep "$PACKAGE_NAME" &> /dev/null

    if [ $? -eq 0 ]; then
      echo "Приложение установлено на $DEVICE"
      break
    else
      echo "Ожидание установки приложения на $DEVICE…"
      sleep 5
    fi
  done
}
```

Скрипт ожидания установки приложения на iOS

```
function waitForAppInstallationIOS {
  local DEVICE_UUID=$1
  local BUNDLE_ID=$2

  while true; do
    installed_apps=$(xcrun simctl listapps "$DEVICE_UUID" 2 > /dev/null | grep "$BUNDLE_ID")

    if [ ! -z "$installed_apps" ]; then
      echo "Приложение установлено на устройстве с UUID: $DEVICE_UUID"
      break
    else
      echo "Ожидание установки приложения на устройстве с UUID: $DEVICE_UUID..."
      sleep 5
    fi
  done
}
```

Скрипт ожидания установки приложения на iOS

```
function waitForAppInstallationIOS {
  local DEVICE_UUID=$1
  local BUNDLE_ID=$2

  while true; do
    installed_apps=$(xcrun simctl listapps "$DEVICE_UUID" 2 > /dev/null | grep "$BUNDLE_ID")

    if [ ! -z "$installed_apps" ]; then
      echo "Приложение установлено на устройстве с UUID: $DEVICE_UUID"
      break
    else
      echo "Ожидание установки приложения на устройстве с UUID: $DEVICE_UUID..."
      sleep 5
    fi
  done
}
```

Скрипт ожидания установки приложения на iOS

```
function waitForAppInstallationIOS {
  local DEVICE_UUID=$1
  local BUNDLE_ID=$2

  while true; do
    installed_apps=$(xcrun simctl listapps "$DEVICE_UUID" 2 > /dev/null | grep "$BUNDLE_ID")

    if [ ! -z "$installed_apps" ]; then
      echo "Приложение установлено на устройстве с UUID: $DEVICE_UUID"
      break
    else
      echo "Ожидание установки приложения на устройстве с UUID: $DEVICE_UUID..."
      sleep 5
    fi
  done
}
```

Запуск тестов на нескольких девайсах

```
#!/bin/bash
source wait_for_installation.sh

DEVICE1="$1"
DEVICE2="$2"
DEVICE3="$3"

PLATFORM="$4"

PACKAGE_NAME="your.package.name"
BUNDLE_ID="com.your.bundle.id"

flutter test integration_test/group_1_test.dart -d "$DEVICE1" & P1=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE1 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE1 $BUNDLE_ID
fi

flutter test integration_test/group_2_test.dart -d "$DEVICE2" & P2=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE2 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE2 $BUNDLE_ID
fi

flutter test integration_test/group_3_test.dart "$DEVICE3" & P3=$!
wait $P1 $P2 $P3
```

Запуск тестов на нескольких девайсах

```
#!/bin/bash
source wait_for_installation.sh

DEVICE1="$1"
DEVICE2="$2"
DEVICE3="$3"

PLATFORM="$4"

PACKAGE_NAME="your.package.name"
BUNDLE_ID="com.your.bundle.id"

flutter test integration_test/group_1_test.dart -d "$DEVICE1" & P1=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE1 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE1 $BUNDLE_ID
fi

flutter test integration_test/group_2_test.dart -d "$DEVICE2" & P2=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE2 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE2 $BUNDLE_ID
fi

flutter test integration_test/group_3_test.dart "$DEVICE3" & P3=$!
wait $P1 $P2 $P3
```

Запуск тестов на нескольких девайсах

```
#!/bin/bash
source wait_for_installation.sh

DEVICE1="$1"
DEVICE2="$2"
DEVICE3="$3"

PLATFORM="$4"

PACKAGE_NAME="your.package.name"
BUNDLE_ID="com.your.bundle.id"

flutter test integration_test/group_1_test.dart -d "$DEVICE1" & P1=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE1 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE1 $BUNDLE_ID
fi

flutter test integration_test/group_2_test.dart -d "$DEVICE2" & P2=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE2 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE2 $BUNDLE_ID
fi

flutter test integration_test/group_3_test.dart "$DEVICE3" & P3=$!
wait $P1 $P2 $P3
```

Запуск тестов на нескольких девайсах

```
#!/bin/bash
source wait_for_installation.sh

DEVICE1="$1"
DEVICE2="$2"
DEVICE3="$3"

PLATFORM="$4"

PACKAGE_NAME="your.package.name"
BUNDLE_ID="com.your.bundle.id"

flutter test integration_test/group_1_test.dart -d "$DEVICE1" & P1=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE1 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE1 $BUNDLE_ID
fi

flutter test integration_test/group_2_test.dart -d "$DEVICE2" & P2=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE2 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE2 $BUNDLE_ID
fi

flutter test integration_test/group_3_test.dart "$DEVICE3" & P3=$!
wait $P1 $P2 $P3
```

Запуск тестов на нескольких девайсах

```
#!/bin/bash
source wait_for_installation.sh

DEVICE1="$1"
DEVICE2="$2"
DEVICE3="$3"

PLATFORM="$4"

PACKAGE_NAME="your.package.name"
BUNDLE_ID="com.your.bundle.id"

flutter test integration_test/group_1_test.dart -d "$DEVICE1" & P1=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE1 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE1 $BUNDLE_ID
fi

flutter test integration_test/group_2_test.dart -d "$DEVICE2" & P2=$!
if [ "$PLATFORM" == "android" ]; then
  waitForAppInstallationAndroid $DEVICE2 $PACKAGE_NAME
elif [ "$PLATFORM" == "ios" ]; then
  waitForAppInstallationIOS $DEVICE2 $BUNDLE_ID
fi

flutter test integration_test/group_3_test.dart "$DEVICE3" & P3=$!
wait $P1 $P2 $P3
```

Пример CI-скрипта

```
integration_test_3_emulators:
  before_script:
    - EMULATOR1=$(bash ./create_emulator.sh ${CI_JOB_ID}_1 “emulator_1”)
    - echo $EMULATOR1
    - bash ./wait_for_emulator_boot.sh “$EMULATOR1”
    - EMULATOR2=$(bash ./create_emulator.sh ${CI_JOB_ID}_2 “emulator_2”)
    - echo $EMULATOR2
    - bash ./wait_for_emulator_boot.sh “$EMULATOR2"
    - EMULATOR3=$(bash ./create_emulator.sh ${CI_JOB_ID}_3 “emulato_3”)
    - echo $EMULATOR3
    - bash ./wait_for_emulator_boot.sh "$EMULATOR3"
  script:
    - bash ./run_3_devices_integration.sh "$EMULATOR1" "$EMULATOR2" "$EMULATOR3" "android"
```

Пример CI-скрипта

```
integration_test_3_emulators:
  before_script:
    - EMULATOR1=$(bash ./create_emulator.sh ${CI_JOB_ID}_1 “emulator_1”)
    - echo $EMULATOR1
    - bash ./wait_for_emulator_boot.sh “$EMULATOR1”
    - EMULATOR2=$(bash ./create_emulator.sh ${CI_JOB_ID}_2 “emulator_2”)
    - echo $EMULATOR2
    - bash ./wait_for_emulator_boot.sh “$EMULATOR2"
    - EMULATOR3=$(bash ./create_emulator.sh ${CI_JOB_ID}_3 “emulato_3”)
    - echo $EMULATOR3
    - bash ./wait_for_emulator_boot.sh "$EMULATOR3"
  script:
    - bash ./run_3_devices_integration.sh "$EMULATOR1" "$EMULATOR2" "$EMULATOR3" "android"
```

Итоги второй части доклада, мы научились:

- Как создать и использовать отдельный Android-эмуляторы и iOS-симуляторы для каждой CI-джобы.

Итоги второй части доклада, мы научились:

- Как создать и использовать отдельный Android-эмуляторы и iOS-симуляторы для каждой CI-джобы.
- Проводить тесты параллельно, не беспокоясь о том, что один тестовый запуск может повлиять на другой.

Итоги второй части доклада, мы научились:

- Как создать и использовать отдельный Android-эмуляторы и iOS-симуляторы для каждой CI-джобы.
- Проводить тесты параллельно, не беспокоясь о том, что один тестовый запуск может повлиять на другой.
- Получили возможность использовать максимально ресурсы железа, запуская несколько девайсов на прогон тестового сьюта.

Подключение и настройка Android Docker и Flutter Docker контейнров

- Использование образа Android-эмулятора

Подключение и настройка Android Docker и Flutter Docker контейнров

- Использование образа Android-эмулятора
- Подготовка Flutter Docker файла

Подключение и настройка Android Docker и Flutter Docker контейнров

- Использование образа Android-эмулятора
- Подготовка Flutter Docker файла
- Кэширование данных

Подключение и настройка Android Docker и Flutter Docker контейнров

- Использование образа Android-эмулятора
- Подготовка Flutter Docker файла
- Кэширование данных
- Запуск тестов в контейнерах с использованием CI

Особенности подготовки Android Docker контейнера:

- Нужно убедиться, что на раннере доступна вертуализация ядра KVM

Особенности подготовки Android Docker контейнера:

- Нужно убедиться, что на раннере доступна вертуализация ядра KVM
- Нужно убедиться, что процессор поддерживает Prescott New Instructions (Оптимальное решение использовать машины с процессорами Intel)

Особенности подготовки Android Docker контейнера:

- Для использования одного контейнера с эмулятором в прогоне, подойдет официальный образы от **Google**

Особенности подготовки Android Docker контейнера:

- Для использования одного контейнера с эмулятором в прогоне, подойдет официальный образы от **Google**
- Для использовать несколько контейнеров с эмуляторами одновременно, удобным решением будет образ от **j-vegas**

Подготовка Flutter Docker файла

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND="noninteractive"
ENV JAVA_VERSION="17"

ENV ANDROID_TOOLS_URL="https://dl.google.com/android/repository/commandlinetools-linux-9477386_latest.zip"
ENV ANDROID_VERSION="33"
ENV ANDROID_BUILD_TOOLS_VERSION="33.0.1"
ENV ANDROID_ARCHITECTURE="x86_64"
ENV ANDROID_SDK_ROOT="/usr/local/android-sdk"

ENV FLUTTER_CHANNEL="stable"
ENV FLUTTER_VERSION="3.10.5"
ENV FLUTTER_URL="https://storage.googleapis.com/flutter_infra_release/releases/$FLUTTER_CHANNEL/linux/flutter_linux_$FLUTTER_VERSION-
  $FLUTTER_CHANNEL.tar.xz"
ENV FLUTTER_ROOT="/opt/flutter"

ENV GRADLE_VERSION="7.4.2"
ENV GRADLE_USER_HOME="/opt/gradle"
ENV GRADLE_URL="https://services.gradle.org/distributions/gradle-${GRADLE_VERSION}-bin.zip"

ENV PATH="$ANDROID_SDK_ROOT/cmdline-tools/latest/bin:$ANDROID_SDK_ROOT/emulator:$ANDROID_SDK_ROOT/platform-tools:
  $ANDROID_SDK_ROOT/platforms:$FLUTTER_ROOT/bin:$GRADLE_USER_HOME/bin:$PATH"
```

Подготовка Flutter Docker файла

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND="noninteractive"
ENV JAVA_VERSION="17"

ENV ANDROID_TOOLS_URL="https://dl.google.com/android/repository/commandlinetools-linux-9477386_latest.zip"
ENV ANDROID_VERSION="33"
ENV ANDROID_BUILD_TOOLS_VERSION="33.0.1"
ENV ANDROID_ARCHITECTURE="x86_64"
ENV ANDROID_SDK_ROOT="/usr/local/android-sdk"

ENV FLUTTER_CHANNEL="stable"
ENV FLUTTER_VERSION="3.10.5"
ENV FLUTTER_URL="https://storage.googleapis.com/flutter_infra_release/releases/$FLUTTER_CHANNEL/linux/flutter_linux_$FLUTTER_VERSION-
  $FLUTTER_CHANNEL.tar.xz"
ENV FLUTTER_ROOT="/opt/flutter"

ENV GRADLE_VERSION="7.4.2"
ENV GRADLE_USER_HOME="/opt/gradle"
ENV GRADLE_URL="https://services.gradle.org/distributions/gradle-${GRADLE_VERSION}-bin.zip"

ENV PATH="$ANDROID_SDK_ROOT/cmdline-tools/latest/bin:$ANDROID_SDK_ROOT/emulator:$ANDROID_SDK_ROOT/platform-tools:
  $ANDROID_SDK_ROOT/platforms:$FLUTTER_ROOT/bin:$GRADLE_USER_HOME/bin:$PATH"
```

Подготовка Flutter Docker файла

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND="noninteractive"
ENV JAVA_VERSION="17"

ENV ANDROID_TOOLS_URL="https://dl.google.com/android/repository/commandlinetools-linux-9477386_latest.zip"
ENV ANDROID_VERSION="33"
ENV ANDROID_BUILD_TOOLS_VERSION="33.0.1"
ENV ANDROID_ARCHITECTURE="x86_64"
ENV ANDROID_SDK_ROOT="/usr/local/android-sdk"

ENV FLUTTER_CHANNEL="stable"
ENV FLUTTER_VERSION="3.10.5"
ENV FLUTTER_URL="https://storage.googleapis.com/flutter_infra_release/releases/$FLUTTER_CHANNEL/linux/flutter_linux_$FLUTTER_VERSION-
  $FLUTTER_CHANNEL.tar.xz"
ENV FLUTTER_ROOT="/opt/flutter"

ENV GRADLE_VERSION="7.4.2"
ENV GRADLE_USER_HOME="/opt/gradle"
ENV GRADLE_URL="https://services.gradle.org/distributions/gradle-${GRADLE_VERSION}-bin.zip"

ENV PATH="$ANDROID_SDK_ROOT/cmdline-tools/latest/bin:$ANDROID_SDK_ROOT/emulator:$ANDROID_SDK_ROOT/platform-tools:
  $ANDROID_SDK_ROOT/platforms:$FLUTTER_ROOT/bin:$GRADLE_USER_HOME/bin:$PATH"
```

Подготовка Flutter Docker файла

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND="noninteractive"
ENV JAVA_VERSION="17"

ENV ANDROID_TOOLS_URL="https://dl.google.com/android/repository/commandlinetools-linux-9477386_latest.zip"
ENV ANDROID_VERSION="33"
ENV ANDROID_BUILD_TOOLS_VERSION="33.0.1"
ENV ANDROID_ARCHITECTURE="x86_64"
ENV ANDROID_SDK_ROOT="/usr/local/android-sdk"

ENV FLUTTER_CHANNEL="stable"
ENV FLUTTER_VERSION="3.10.5"
ENV FLUTTER_URL="https://storage.googleapis.com/flutter_infra_release/releases/$FLUTTER_CHANNEL/linux/flutter_linux_$FLUTTER_VERSION-
  $FLUTTER_CHANNEL.tar.xz"
ENV FLUTTER_ROOT="/opt/flutter"

ENV GRADLE_VERSION="7.4.2"
ENV GRADLE_USER_HOME="/opt/gradle"
ENV GRADLE_URL="https://services.gradle.org/distributions/gradle-${GRADLE_VERSION}-bin.zip"

ENV PATH="$ANDROID_SDK_ROOT/cmdline-tools/latest/bin:$ANDROID_SDK_ROOT/emulator:$ANDROID_SDK_ROOT/platform-tools:
  $ANDROID_SDK_ROOT/platforms:$FLUTTER_ROOT/bin:$GRADLE_USER_HOME/bin:$PATH"
```

Подготовка Flutter Docker файла

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND="noninteractive"
ENV JAVA_VERSION="17"

ENV ANDROID_TOOLS_URL="https://dl.google.com/android/repository/commandlinetools-linux-9477386_latest.zip"
ENV ANDROID_VERSION="33"
ENV ANDROID_BUILD_TOOLS_VERSION="33.0.1"
ENV ANDROID_ARCHITECTURE="x86_64"
ENV ANDROID_SDK_ROOT="/usr/local/android-sdk"

ENV FLUTTER_CHANNEL="stable"
ENV FLUTTER_VERSION="3.10.5"
ENV FLUTTER_URL="https://storage.googleapis.com/flutter_infra_release/releases/$FLUTTER_CHANNEL/linux/flutter_linux_$FLUTTER_VERSION-
  $FLUTTER_CHANNEL.tar.xz"
ENV FLUTTER_ROOT="/opt/flutter"

ENV GRADLE_VERSION="7.4.2"
ENV GRADLE_USER_HOME="/opt/gradle"
ENV GRADLE_URL="https://services.gradle.org/distributions/gradle-${GRADLE_VERSION}-bin.zip"

ENV PATH="$ANDROID_SDK_ROOT/cmdline-tools/latest/bin:$ANDROID_SDK_ROOT/emulator:$ANDROID_SDK_ROOT/platform-tools:
  $ANDROID_SDK_ROOT/platforms:$FLUTTER_ROOT/bin:$GRADLE_USER_HOME/bin:$PATH"
```

Подготовка Flutter Docker файла

```
# Устанавливаем необходимые зависимости
RUN apt-get update \
  && apt-get install --yes --no-install-recommends \
    openjdk-$JAVA_VERSION-jdk \
    curl \
    unzip \
    sed \
    git \
    bash \
    xz-utils \
    libglvnd0 \
    ssh \
    xauth \
    x11-xserver-utils \
    libglu1 \
    libx11-6 libxcb1 libxdamage1 libnss3 libxcursor1 libxi6 libxext6 libxfixes3 \
    lib32stdc++6 \
    libpulse0 \
    libxcomposite1 \
    libgl1-mesa-glx \
    clang \
    cmake \
    ninja-build \
    pkg-config \
    libgtk-3-dev \
  && rm -rf /var/lib/{apt,dpkg,cache,log}
```

Подготовка
Flutter Docker файла

```
# Устанавливаем Gradle.
RUN curl -L $GRADLE_URL -o gradle-$GRADLE_VERSION-bin.zip \
  && apt-get install -y unzip \
  && unzip gradle-$GRADLE_VERSION-bin.zip \
  && mv gradle-$GRADLE_VERSION $GRADLE_USER_HOME \
  && rm gradle-$GRADLE_VERSION-bin.zip
```

Подготовка
Flutter Docker файла

```
# Устанавливаем Android SDK.
RUN mkdir /root/.android \
  && touch /root/.android/repositories.cfg \
  && mkdir -p $ANDROID_SDK_ROOT \
  && curl -o android_tools.zip $ANDROID_TOOLS_URL \
  && unzip -qq -d $ANDROID_SDK_ROOT android_tools.zip \
  && rm android_tools.zip \
  && mv $ANDROID_SDK_ROOT/cmdline-tools $ANDROID_SDK_ROOT/latest \
  && mkdir -p $ANDROID_SDK_ROOT/cmdline-tools \
  && mv $ANDROID_SDK_ROOT/latest $ANDROID_SDK_ROOT/cmdline-tools/latest \
  && yes "y" | sdkmanager "build-tools;$ANDROID_BUILD_TOOLS_VERSION" \
  && yes "y" | sdkmanager "platforms;android-$ANDROID_VERSION" \
  && yes "y" | sdkmanager "platform-tools"
```

Подготовка
Flutter Docker файла

```
# Устанавливаем Android SDK.
RUN mkdir /root/.android \
 && touch /root/.android/repositories.cfg \
 && mkdir -p $ANDROID_SDK_ROOT \
 && curl -o android_tools.zip $ANDROID_TOOLS_URL \
 && unzip -qq -d $ANDROID_SDK_ROOT android_tools.zip \
 && rm android_tools.zip \
 && mv $ANDROID_SDK_ROOT/cmdline-tools $ANDROID_SDK_ROOT/latest \
 && mkdir -p $ANDROID_SDK_ROOT/cmdline-tools \
 && mv $ANDROID_SDK_ROOT/latest $ANDROID_SDK_ROOT/cmdline-tools/latest \
 && yes "y" | sdkmanager "build-tools;$ANDROID_BUILD_TOOLS_VERSION" \
 && yes "y" | sdkmanager "platforms;android-$ANDROID_VERSION" \
 && yes "y" | sdkmanager "platform-tools"
```

Подготовка
Flutter Docker файла

```
# Устанавливаем Flutter.
RUN curl -o flutter.tar.xz $FLUTTER_URL \
  && mkdir -p $FLUTTER_ROOT \
  && tar xf flutter.tar.xz -C /opt/ \
  && rm flutter.tar.xz \
  && git config --global --add safe.directory /opt/flutter \
  && flutter config --no-analytics \
  && flutter precache \
  && yes "y" | flutter doctor --android-licenses \
  && flutter doctor \
  && flutter update-packages
```

Запуск тестов
в Docker контейнерах

```
integration-test-in-adroid-docker:
  image: registry.your.url/flutter
  services:
    - name: registry.your.url/android-emulator
      alias: emulator
      command: ["/entrypoint.sh","5554","5555"]
  before_script:
    - adb connect emulator:5555
    - chmod +x ./wait_for_emulator_boot.sh
    - bash ./wait_for_emulator_boot.sh emulator:5555
    - flutter devices
script:
  - flutter test integration_test/your_test.dart -d emulator:5555
```

Запуск тестов
в Docker контейнерах

```
integration-test-in-adroid-docker:
  image: registry.your.url/flutter
  services:
    - name: registry.your.url/android-emulator
      alias: emulator
      command: ["/entrypoint.sh","5554","5555"]
  before_script:
    - adb connect emulator:5555
    - chmod +x ./wait_for_emulator_boot.sh
    - bash ./wait_for_emulator_boot.sh emulator:5555
    - flutter devices
script:
  - flutter test integration_test/your_test.dart -d emulator:5555
```

Запуск тестов в Docker контейнерах

```
#!/usr/bin/env bash

# Скрипт для перенаправления всего трафика на подключение adb

# Определяем IP и перенаправляем порты ADB наружу на внешний интерфейс
ip=$(ifconfig eth0 | grep 'inet' | cut -d: -f2 | awk '{ print $2}')

# Подключаемся к локальному TCP-порту 5037 и прослушивае команды
socat tcp-listen:5037,bind=$ip,fork tcp:127.0.0.1:5037 &

# Каждый эмулятор использует пару последовательных портов
# Порт с четным номером для консольных подключений
socat tcp-listen:$1,bind=$ip,fork tcp:127.0.0.1:5554 &

# Порт с нечетным номером для adb подключений
socat tcp-listen:$2,bind=$ip,fork tcp:127.0.0.1:5555 &
```

Запуск тестов в Docker контейнерах

```
#!/usr/bin/env bash

# Скрипт для перенаправления всего трафика на подключение adb

# Определяем IP и перенаправляем порты ADB наружу на внешний интерфейс
ip=$(ifconfig eth0 | grep 'inet' | cut -d: -f2 | awk '{ print $2}')

# Подключаемся к локальному TCP-порту 5037 и прослушивае команды
socat tcp-listen:5037,bind=$ip,fork tcp:127.0.0.1:5037 &

# Каждый эмулятор использует пару последовательных портов
# Порт с четным номером для консольных подключений
socat tcp-listen:$1,bind=$ip,fork tcp:127.0.0.1:5554 &

# Порт с нечетным номером для adb подключений
socat tcp-listen:$2,bind=$ip,fork tcp:127.0.0.1:5555 &
```

Запуск тестов в Docker контейнерах

```
#!/usr/bin/env bash

# Скрипт для перенаправления всего трафика на подключение adb

# Определяем IP и перенаправляем порты ADB наружу на внешний интерфейс
ip=$(ifconfig eth0 | grep 'inet' | cut -d: -f2 | awk '{ print $2}')

# Подключаемся к локальному TCP-порту 5037 и прослушивае команды
socat tcp-listen:5037,bind=$ip,fork tcp:127.0.0.1:5037 &

# Каждый эмулятор использует пару последовательных портов
# Порт с четным номером для консольных подключений
socat tcp-listen:$1,bind=$ip,fork tcp:127.0.0.1:5554 &

# Порт с нечетным номером для adb подключений
socat tcp-listen:$2,bind=$ip,fork tcp:127.0.0.1:5555 &
```

Запуск тестов
в Docker контейнерах

```
#!/usr/bin/env bash

# Скрипт для перенаправления всего трафика на подключение adb

# Определяем IP и перенаправляем порты ADB наружу на внешний интерфейс
ip=$(ifconfig eth0 | grep 'inet' | cut -d: -f2 | awk '{ print $2}')

# Подключаемся к локальному TCP-порту 5037 и прослушивае команды
socat tcp-listen:5037,bind=$ip,fork tcp:127.0.0.1:5037 &

# Каждый эмулятор использует пару последовательных портов
# Порт с четным номером для консольных подключений
socat tcp-listen:$1,bind=$ip,fork tcp:127.0.0.1:5554 &

# Порт с нечетным номером для adb подключений
socat tcp-listen:$2,bind=$ip,fork tcp:127.0.0.1:5555 &
```

Запуск тестов
в Docker контейнерах

```
integration-test-in-adroid-docker:
  image: registry.your.url/flutter
  services:
    - name: registry.your.url/android-emulator
      alias: emulator
      command: ["/entrypoint.sh","5554","5555"]
  before_script:
    - adb connect emulator:5555
    - chmod +x ./wait_for_emulator_boot.sh
    - bash ./wait_for_emulator_boot.sh emulator:5555
    - flutter devices
  script:
    - flutter test integration_test/your_test.dart -d emulator:5555
```

Запуск тестов
в Docker контейнерах

```
integration-test-in-adroid-docker:
  image: registry.your.url/flutter
  services:
    - name: registry.your.url/android-emulator
      alias: emulator
      command: ["/entrypoint.sh","5554","5555"]
  before_script:
    - adb connect emulator:5555
    - chmod +x ./wait_for_emulator_boot.sh
    - bash ./wait_for_emulator_boot.sh emulator:5555
    - flutter devices
script:
  - flutter test integration_test/your_test.dart -d emulator:5555
```

Будут ли проходить тесты?

Кэширование данных

/root/.pub-cache",
/opt/gradle",
/usr/local/android-sdk/build-tools",
/usr/local/android-sdk/emulator",
/usr/local/android-sdk/patcher",
/usr/local/android-sdk/platforms",
/usr/local/android-sdk/tools"

Кэширование данных

```
root@runner:~# cat /etc/gitlab-runner/config.toml
[[runners]]
  [runners.docker]
    volumes = ["/builds:/builds", "/cache_mobileapp:/root/.pub-cache", "/cache_mobileapp_gradle:/
opt/gradle", "/cache_mobileapp_android/build-tools:/usr/local/android-sdk/build-tools", "/
cache_mobileapp_android/emulator:/usr/local/android-sdk/emulator", "/cache_mobileapp_android/
patcher:/usr/local/android-sdk/patcher", "/cache_mobileapp_android/platforms:/usr/local/android-
sdk/platforms", "/cache_mobileapp_android/tools:/usr/local/android-sdk/tools"]
```

Запуск тестов
в Docker контейнерах

```
import 'dart:async';

import 'package:integration_test/integration_test_driver.dart';

Future<void> main() async => integrationDriver();
```

Запуск тестов
в Docker контейнерах

```
integration-test-in-adroid-docker:
  image: registry.your.url/flutter
  services:
    - name: registry.your.url/android-emulator
      alias: emulator
      command: ["/entrypoint.sh","5554","5555"]
  before_script:
    - adb connect emulator:5555
    - chmod +x ./wait_for_emulator_boot.sh
    - bash ./wait_for_emulator_boot.sh emulator:5555
    - flutter devices
script:
  - flutter drive --driver=test_driver/integration_driver.dart --target=integration_test/your_test.dart -d emulator:5555
```

Запуск тестов
в Docker контейнерах

```
integration-test-in-adroid-docker:
  image: registry.your.url/flutter
  services:
    - name: registry.your.url/android-emulator
      alias: emulator
      command: ["/entrypoint.sh","5554","5555"]
  before_script:
    - adb connect emulator:5555
    - chmod +x ./wait_for_emulator_boot.sh
    - bash ./wait_for_emulator_boot.sh emulator:5555
    - flutter devices
script:
  - flutter test integration_test/your_test.dart -d emulator:5555
```

Итоги CI части доклада, мы научились:

- Мы научились динамически создавать и дожидаться полной загрузки эмулятора, и установки тестового приложения перед запуском тестов.

Итоги CI части доклада, мы научились:

- Мы научились динамически создавать и дожидаться полной загрузки эмулятора, и установки тестового приложения перед запуском тестов.
- Оптимизировать процесс билда тестового приложения в Docker контейнерах используя кэширование.

Итоги CI части доклада, мы научились:

- Мы научились динамически создавать и дожидаться полной загрузки эмулятора, и установки тестового приложения перед запуском тестов.
- Оптимизировать процесс билда тестового приложения в Docker контейнерах используя кэширование.
- Возможность запуска тестов на нескольких устройствах параллельно.

Подведем итоги доклада:

- Процесс тестирования на Flutter имеет свои ограничения, но при правильном подходе его можно сделать эффективным.

Подведем итоги доклада:

- Процесс тестирования на Flutter имеет свои ограничения, но при правильном подходе его можно сделать эффективным.
- Особенное внимание стоит уделить, инициализации и запуску тестов, что позволит ускорить время тестового прогона в десятки раз.

Материалы доклада

С удовольствие ответим на ваши вопросы

Ахмат Султанов
Спикер доклада

Роман Плужников
Приглашённый эксперт