БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики
Кафедра технологий программирования

Лабораторная работа №2
Коммуникация в многоконтейнерных системах

По дисциплине «Непрерывное интегрирование и сборка программного обеспечения»

Методические указания по выполнению лабораторной работы

Подготовила:
Давидовская М. И.,
Ст. преподаватель кафедры ТП

Минск, 2025 г.

Содержание

Цель работы

Проектирование распределенных систем и реализация коммуникации между их компонентами

Задачи работы

1. Реализовать простую распределенную систему
2. Реализовать распределенную систему с асинхронным обменом на основе брокера сообщений
3. Оформить отчёт в формате Markdown и опубликовать результаты.

Краткие теоретические сведения

Издатель (publisher) или «производитель» (producer) — это приложение (или экземпляр приложения), которое публикует (создает) сообщения. Одно и то же приложение может также использовать сообщения и, таким образом, одновременно быть потребителем (consumer).

Термин «потребитель» (Consumer) в разных предметных областях может иметь разные значения. В общем, в контексте обмена сообщениями и потоковой передачи данных **потребитель** — это приложение (или экземпляр приложения), которое использует и подтверждает сообщения. Одно и то же приложение может также публиковать сообщения и, таким образом, одновременно быть «издателем» (publisher).

В протоколах обмена сообщениями используется концепция постоянной подписки на доставку сообщений. **Подписка** — это один из терминов, обычно используемых для описания такой сущности. **Потребитель** (consumer) — другой ключевой термин. В протоколах обмена сообщениями, поддерживаемых RabbitMQ, используются оба термина, но в документации RabbitMQ предпочтение отдается последнему.

Потребитель — это подписка на рассылку сообщений, которая должна быть зарегистрирована до начала рассылки и может быть отменена приложением.

RabbitMQ — это посредник обмена сообщениями. Он принимает сообщения от издателей, маршрутизирует их и, если есть очереди для

маршрутизации, сохраняет их для использования или немедленно доставляет потребителям при их наличии.

Издатели публикуют сообщения по назначению, которое зависит от протокола. В AMQP 0-9-1 издатели публикуют материалы в точках обмена (exchange). В AMQP 1.0 публикация осуществляется по ссылке (link). В MQTT издатели публикуют материалы в разделах (topics). Наконец, STOMP поддерживает множество типов адресатов: разделы, очереди, точки обмена по протоколу AMQP 0-9-1.

Опубликованное сообщение должно быть перенаправлено в очередь (раздел и т.д.). В очереди (разделе) могут быть онлайн потребители. Когда сообщение перенаправлено в очередь и в сети есть потребитель, который может принимать доставку, сообщение будет отправлено получателю.

Попытка публикации в несуществующей очереди (теме) приведет к возникновению исключения с кодом 404.

Потребители получают сообщения из очередей. При добавлении нового потребителя, при условии, что в очереди уже есть готовые сообщения, доставка начнется немедленно.

Целевая очередь может быть пустой на момент регистрации потребителя. И в таком случае первые доставки будут осуществляться при постановке новых сообщений в очередь.

Задания

Методические указания

Все результаты лабораторной работы должны быть опубликованы в git-репозитории, ссылка на который доступна в курсе «Непрерывное интегрирование и сборка программного обеспечения»

Критерии оценивания

Для групп 11-13 выполнить задания №1-2 (оба задания), для группы 14 — задание 1 и упражнение 2.2 из задания №2.

Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Код приложений, конфигурационных файлов.

4. Ответы на контрольные вопросы.

Отчет должен быть опубликован в git-репозитории на github. Все результаты лабораторной работы должны быть опубликованы в git-репозитории, ссылка на который доступна в курсе «Непрерывное интегрирование и сборка программного обеспечения».

В файле Readme проекта на github должна быть ссылка на отчёт. Отчет опубликовать во внешнем хранилище или в репозитории в каталоге /docs. Если в лабораторной работе необходимо написать программу/ы, то отчёт должен результаты тестов по каждой программе и ответы на контрольные вопросы.

Пример оформления файла Readme может быть таким:

```
# Overview

Report on LabRabota2.

# Usage

// Заменить <<link>> и <<folder>> на соответствующие ссылки/названия
To check, please, preview report by <<link>> and source files
in <<folder>>.

# Author

Your name and group number.

# Additional Notes

// СКОПИРОВАТЬ И ВСТАВИТЬ ССЫЛКУ НА СВОЙ РЕПОЗИТОРИЙ,НАПРИМЕР
https://github.com/maryiad/lab3-task1-gr16-david
```

Каждая лабораторная работа содержит тексты задач и контрольные вопросы, ответы на которые проверяются преподавателем при приёме работы у студента.

Выполнение студентом лабораторной работы и сдача её результатов преподавателю происходит следующим образом:

1. Студент выполняет разработку программ.

2. В ходе разработки студент обязан следовать указаниям к данной задаче (в случае их наличия). Исходные тексты программ следует разрабатывать в соответствии с требованиями к оформлению, приведёнными в приложении.

3. Студент выполняет самостоятельную проверку исходного текста каждой разработанной программы и правильности её работы, а также свои знания по теме лабораторной работы.

Задание 1. Разработка простейшего распределенного приложения

1. Изучить материалы по монтированию каталогов (связанных томов монтирования — **bind mounts**) из хост-системы в контейнер docker:
   - https://docs.docker.com/engine/storage/ — типы хранилищ;
   - https://docs.docker.com/engine/storage/bind-mounts/ — монтирование связанных томов (bind mounts).
2. Согласно вашему варианту разработать два приложения так, чтобы результат первого являлся исходными данными для второго. Язык разработки может быть любым, например Python, или C/C++, или Java, или другой.
3. В каталоге репозитория для лабораторной работы 2 создать ветку **task1** и каталог **task1**. Каталог **task1** должен содержать файлы **Readme**, **.gitignore**, **.dockerignore**, **docker-compose.yml**.
4. В каталоге **task1** создать каталоги **data**, **result-1**, **result, worker1**, **worker2**.
5. Для связи двух приложений использовать следующую схему, где символ «**.**» означает путь относительно текущего каталога, т. е. **task1**:
   - Каталог **./data** должен быть примонтирован в каталог **/var/data** для программы 1. — *Оттуда будут браться исходные данные.*
   - Каталог **./result-1** должен быть примонтирован в каталог /var/result для программы 1. — *Туда будут складываться промежуточные данные.*
   - Каталог **./result-1** должен быть примонтирован в каталог **/var/data** для программы 2. — *Оттуда будут браться промежуточные результаты.*
   - Каталог **./result** должен быть примонтирован в каталог **/var/result** для программы 2. — *Туда будут складывать результаты финальной обработки.*
6. В каталогах **worker1**, **worker2** должны содержаться исходный файлы приложения, файл **.gitignore** (при необходимости), и

конфигурационный файл Dockerfile для сборки образов. Dockerfile должен содержать комментарии к блокам инструкций.

7. Собрать образы docker для каждого из созданных приложений, описав конфигурацию в Dockerfile.
8. Создать файл **docker-compose.yml** в каталоге **task1** для сборки и запуска приложений с помощью команды

```
docker compose up –build
```

Требования к файлу **docker-compose.yml**:

- Два сервиса.
- Каждый соответствует программам согласно варианту.
- Объявлена директива **build** для каждого сервиса.
- Монтирование каталогов с данными.
- Описание зависимостей сервисов одного от другого.

Варианты к заданию 1

1. **Вариант задания соответствует порядковому номеру в группе**.
2. Содержимое исходных файлов — целые числа.
3. Результат выполнения программы 2 необходимо дополнительно вывести на экран.

Варианты для программ 1 и 2

№	**Программа 1**	**Программа 2**
1	Ищет в каталоге **/var/data** самый большой по объёму файл и перекладывает его в **/var/result/data.txt**	Сохраняет произведение первого и последнего числа из файла **/var/data/data.txt** в **/var/result/result.txt**
2	Ищет в каталоге **/var/data** файл с наибольшим количеством строк и перекладывает его в **/var/result/data.txt**	Ищет наибольшее число из файла **/var/data/data.txt** и сохраняет его вторую степень в **/var/result/result.txt**

3	Формирует файл **/var/result/data.txt** из первых строк всех файлов каталога **/var/data**	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет его третью степень в **/var/result/result.txt**
4	Формирует файл **/var/result/data.txt** так, что каждая строка файла - количество строк в файлах из каталога **/var/data**.	Ищет наибольшее число из файла **/var/data/data.txt** и сохраняет количество таких чисел из последовательности в **/var/result/result.txt**.
5	Формирует файл **/var/result/data.txt** так, что каждая строка файла - количество символов в именах файлов из каталога **/var/data**.	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет количество таких чисел из последовательности в **/var/result/result.txt**
6	Ищет в каталоге **/var/data** файл с самым коротким названием и перекладывает его в **/var/result/data.txt**.	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет его третью степень в **/var/result/result.txt**
7	Берёт из каталога **/var/data** случайный файл и перекладывает его в **/var/result/data.txt**.	Ищет наибольшее число из файла **/var/data/data.txt** и сохраняет количество таких чисел из последовательности в **/var/result/result.txt**.
8	Ищет в каталоге **/var/data** самый большой по объёму файл и перекладывает его в **/var/result/data.txt**	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет количество таких чисел из последовательности в **/var/result/result.txt**
9	Ищет в каталоге **/var/data** файл с	Ищет наибольшее число из

	наибольшим количеством строк и перекладывает его в **/var/result/data.txt**	файла **/var/data/data.txt** и сохраняет его вторую степень в **/var/result/result.txt**
10	Формирует файл **/var/result/data.txt** из первых строк всех файлов каталога **/var/data**	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет его третью степень в **/var/result/result.txt**
11	Формирует файл **/var/result/data.txt** так, что каждая строка файла - количество строк в файлах из каталога **/var/data**.	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет его третью степень в **/var/result/result.txt**
12	Формирует файл **/var/result/data.txt** так, что каждая строка файла - количество символов в именах файлов из каталога **/var/data**.	Ищет наименьшее число из файла **/var/data/data.txt** и сохраняет количество таких чисел из последовательности в **/var/result/result.txt**
13	Ищет в каталоге **/var/data** файл с самым коротким названием и перекладывает его в **/var/result/data.txt**.	Сохраняет произведение первого и последнего числа из файла **/var/data/data.txt** в **/var/result/result.txt**
14	Берёт из каталога **/var/data** случайный файл и перекладывает его в **/var/result/data.txt**.	Ищет наибольшее число из файла **/var/data/data.txt** и сохраняет его вторую степень в **/var/result/result.txt**

Задание 2. Асинхронная коммуникация в распределенных системах на основе брокера сообщений

1. В репозитории создать ветку **task2** и каталоги **task2-1**, **task2-2**, **task2-3** для каждого из упражнений.
2. Выполнить упражнения 2.1, 2.2 и.2.3.
3. Опубликовать исходный код приложений, конфигурационных файлов **Dockerfile**, **docker-compose.yml** в репозиторий.
4. Продемонстрировать использование .gitignore, .dockerignore.
5. Оформить файл **Readme** для ветки **task2**.

Упражнение 2.1

1. Изучить статью RabbitMQ для аналитика: практический ликбез.
2. Создать учетную запись на сервисе CloudAMQP.
3. Реализовать пример из статьи RabbitMQ для аналитика: практический ликбез.
4. В каталоге **task2-1** опубликовать фрагменты json и описание хода выполнения задания.

Упражнение 2.2

1. Изучить статью Setup RabbitMQ with Docker Compose.
2. Настроить контейнер докер и продемонстрировать получение доступа к интерфейсу администрирования.
3. Пройти уроки 1, 2 и 3 из RabbitMQ Tutorials. **Группы 11-13** — на любом языке программирования, **группа 14** — язык программирования Python. Код из уроков хранится локально на хост-системе. Контейнер создавать не надо.
4. При защите лабораторной работы продемонстрировать работу брокера сообщений на основе примеров из уроков 1-3.

Упражнение 2.3

1. Изучить статьи:
   - Использование RabbitMQ в Python: от Docker до распределённых систем.
   - Как запускать RabbitMQ в Docker.
   - Installing and using RabbitMQ with Docker.
   - Пишем продюсер и консумер для RabbitMQ на Python в Google Colab.

2. Настроить распределенную систему, состоящую из контейнеров с Rabbit-MQ, Producer (приложение публикует сообщения в очередь) и Consumer (приложение читает из очереди), взяв за основу пример из любой статьи.
3. В качестве хранилища в каждом приложении использовать связанную точку монтирования (bind mount).
4. Для каждого из контейнеров создать Dockerfile и для управления инфраструктурой настроить файл docker-compose.yml.
5. Продемонстрировать использование консольных утилит rabbitmqctl, rabbitmqadmin.
6. Опубликовать конфигурационные файлы и файлы с исходным кодом приложений.

Контрольные вопросы

1. Что такое обмен сообщениями (messaging).
2. Что такое брокер сообщений?
3. Дайте определение RabbitMQ?
4. Какой прототокол лежит в основе RabitMQ?
5. В RabbitMQ сообщения, принятые от приложения-продюсера (издателя), записываются …...
6. Если необходимо просто распараллелить обработку сообщений, принятых от продюсера (издателя) по нескольким потребителям, какой тип обменника надо выбрать в RabbitMQ?
7. Обычно приложение-продюсер отправляя сообщение в RabbitMQ, отправляет его ……
8. Как предотвратить потребление устаревших данных из RabbitMQ?
9. По какому принципу LIFO или FIFO устроены очереди в RabbitMQ, в которые записываются сообщения?
10. Что такое Binding в RabbitMQ?
11. Дайте определение Routing Key в Rabbit MQ.
12. Объясните, что такое точка обмена (exchange).
13. Опишите по шагам процесс работы RabbitMQ
14. Для чего предназначен Server в RabbitMQ.
15. Опишите назначение Vhost в RabbitMQ.