|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Министерство науки и высшего образования РФ | | | | | | | | |  | |
|  | | | | |  | | |  | | | | |
|  | | | | ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | |  | | | |
|  | | | | |  | | |  | | | | |
|  | | | **Технологии разработки распределенных приложений**  Лабораторная работа №4  «Разработка распределенного приложения для локальной сети» | | | | | | |  | | |
|  | | | | |  | | |  | | | | |
|  | Работу выполнили  Студенты гр. ПМИ-4-18:  Колесников А.С  Пухов Н.А. | | | | |  | Проверил  преподаватель кафедры МОВС  Постаногов И.С  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г. | | | | |  | |
|  | | | | |  | | |  | | | | |
|  | | | | | Пермь 2021 | | |  | | | | |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc91519968)

[2 Архитектура системы 3](#_Toc91519969)

[3 Архитектура модулей системы 5](#_Toc91519970)

[3.1 Архитектура диспетчера 5](#_Toc91519971)

[3.2 Архитектура рабочего сервера 5](#_Toc91519972)

[3.3 Архитектура клиента 5](#_Toc91519973)

[4 Методы коммуникации компонентов системы 7](#_Toc91519974)

[5 Способ передачи данных 7](#_Toc91519975)

[6 Структура передаваемых данных 7](#_Toc91519976)

[7 Отказоустойчивость системы 9](#_Toc91519977)

[8 Тестирование с помощью Chaos Framework 9](#_Toc91519978)

[9 Исходный код приложения 9](#_Toc91519979)

[9.1 Клиент 9](#_Toc91519980)

[9.2 Сервер-диспетчер 17](#_Toc91519981)

[9.3 Сервер-документа 18](#_Toc91519982)

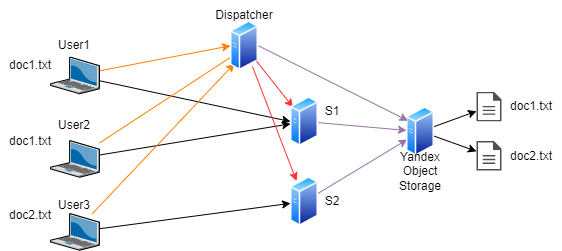
1. Постановка задачи

Задача заключается в создании распределенного приложения по совместному редактированию документов. Пользователи могут одновременно вносить свои изменения в один или несколько документов и моментально видеть изменения документа, которые производятся другими пользователями. При этом приложением должны разрешаться конфликтные ситуации, когда несколько пользователей пытаются изменить один и тот же фрагмент текста.

1. Архитектура системы

Архитектура приложения является сервис-ориентированной и состоит из одного диспетчера, одного или нескольких серверов для работы с одним документом, а также облачного файлового хранилища от Yandex. Приложением может пользоваться один или несколько клиентов.

1. Диспетчер (Dispatcher). В функции диспетчера входит предоставление клиентам информации о списке доступных документов и о том, какой сервер работает с тем или иным документом.
2. Рабочий сервер (S1, S2). В обязанности рабочего сервера входит предоставление услуг клиентам по работе с одним документом: получение содержимого документа актуальной версии, прием и отправка сделанных изменений, а также сохранение новой версии документа.



Рисунок

1 – Архитектура системы

Выбор архитектуры приложения продиктован необходимостью иметь состояние на рабочих серверах, так как совместное редактирование документов предполагает наличие на серверах объектов документов в оперативной памяти, которые должны содержать информацию о том, какие изменения относительно какой версии документа были сделаны.

На рисунке 1.2 представлена архитектура развертывания на кластере под управлением Kubernetes. Системой тестирования Chaos Framework требовалось использование объектов только типа Pod или Deployment.

Доступ к тому или иному сервису осуществляется при помощи указания доменного имени trrp.mooo.com и порта сервиса:

* 30168 – диспетчер
* 30001 – документный сервер 1
* 30002 – документный сервер 2
* … (при желании можно легко добавить новые документные серверы)

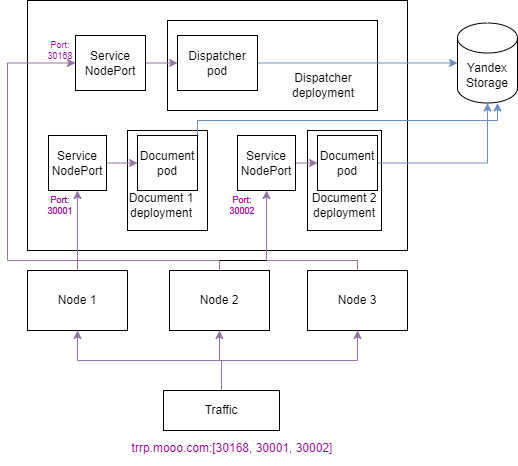


Рисунок 1.2 – Архитектура развертывания на Kubernetes

Стоит отметить, что данная архитектура позволяет беспрепятственно добавить новые экземпляры диспетчера. В таком случае клиентские запросы будут проксированы на случайный диспетчер. Это позволит повысить отказоустойчивость системы в целом.

1. Архитектура модулей системы

Серверная часть была реализована на языке Python, клиентская часть на языке C# с применением технологии WPF.

* 1. Архитектура диспетчера

Диспетчер предоставляет клиентам информации о том, какой сервер работает с тем или иным документом. Это позволяет сделать балансировку нагрузки между несколькими рабочими серверами, тем самым давая возможность горизонтального масштабирования. Предоставляемая диспетчером информация состоит из IP-адреса рабочего сервера. Также диспетчер предоставляет команды для получения списка доступных документов, делая запрос к облачному хранилищу от Yandex. Диспетчер многопоточен, то есть умеет обрабатывать параллельно несколько клиентских запросов.

* 1. Архитектура рабочего сервера

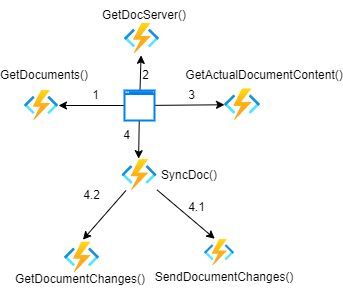
В обязанности рабочего сервера входит предоставление услуг клиентам по работе с одним документом: получение содержимого документа актуальной версии, прием и отправка сделанных изменений, сохранение новых версии документа в Yandex Object Storage. Многопоточность присутствует на уровне редактирования нескольких документов.

* 1. Архитектура клиента

Алгоритм работы клиента:

1. Происходит запуск графического приложения, которое подключается к серверу-диспетчеру и получает список всех доступных для редактирования документов;
2. После выбора пользователем нужного документа, отправляет запрос на сервер-диспетчер для получения адреса рабочего сервера;
3. Клиент подкачается к рабочему серверу с документом и получает актуальную версию документа;
4. Происходит синхронизация версии на клиенте с версией на сервере с периодичностью 500 мс:
   1. Вычисляются различия в версиях клиента и сервера, которые после отправляются серверу;
   2. Клиент запрашивает изменения с сервера. Если изменения имеются, то применяет их к своей версии документа.

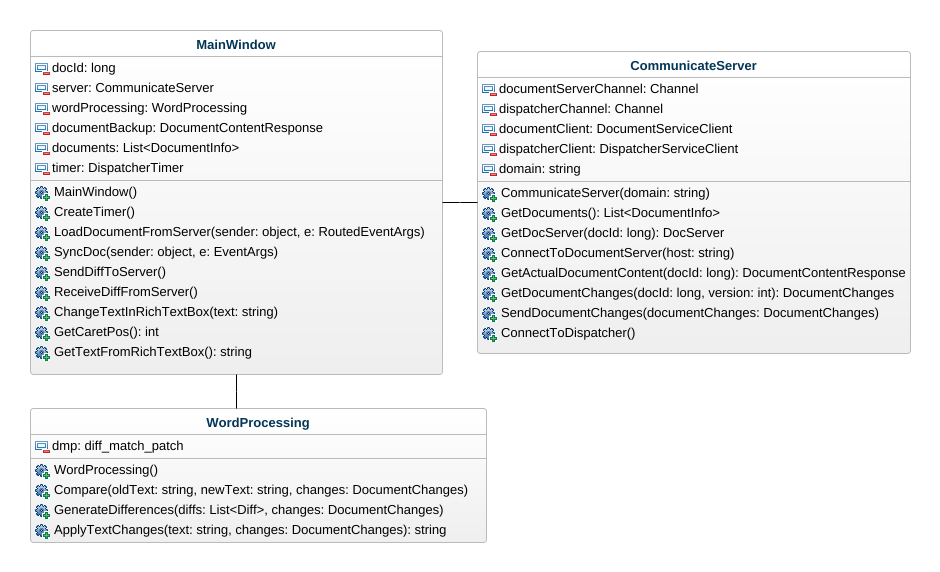
Данный алгоритм изображен на рисунке 2.



Рисунок

2 – Архитектура клиента

Также была составлена диаграмма классов (рис. 3), где класс MainWindow класс главного окна графического приложения, который содержит все обработчики событий. Класс CommunicateServer отвечает за коммуникацию с серверами. Класс WordProcessing выделяет и применяет изменения к тексту.



Рисунок

3 – Диаграмма классов клиента

1. Методы коммуникации компонентов системы

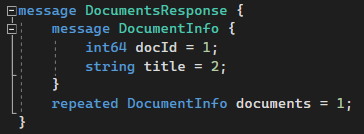
Для коммуникации клиента с сервером была выбрана система удаленного вызова процедур gRPC, т.к. она предоставляет все необходимые инструменты, которые позволяют надежно и эффективно выстраивать коммуникацию между несколькими серверами в контексте сервис-ориентированной архитектуры.

1. Способ передачи данных

Поскольку используется технология gRPC, которая является одной из реализаций удаленного вызова процедур, то способ передачи данных - нерезидентная синхронная связь с синхронизацией по ответу, так как ответ приходит клиенту только тогда, когда сообщение обработано на сервере, при этом сообщение нигде не сохраняется в отличие от очереди сообщений.

1. Структура передаваемых данных

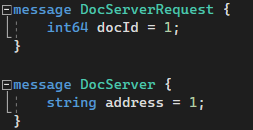
Так как методом коммуникации была выбрана система gRPC, то структуры передаваемых данных описывались в proto-файлах. Для получения информации о существующих файлах отправляется запрос диспетчеру, который возвращает список документов, где каждый документ имеет id документа и название (рис. 4).



Рисунок

4 – Список документов, возвращаемый сервером-хранилищем

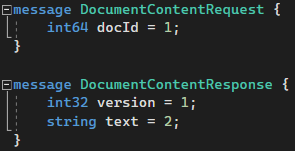
Для получения рабочего сервера, на диспетчер отправляется запрос с нужным id, ответом на который является адрес рабочего сервера (рис. 5).



Рисунок

5 – Запрос и ответ сервера-диспетчера

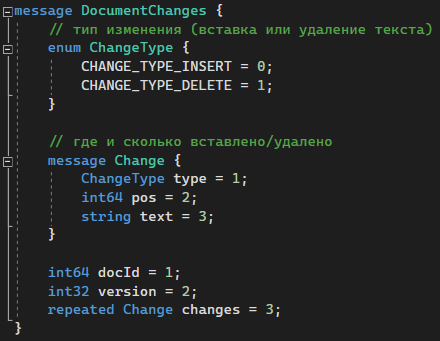
Для получения содержимого документа создается запрос с id-документа, ответом на который приходит содержимое документа и его версия (рис. 6).



Рисунок

6 – Получение содержимого документа

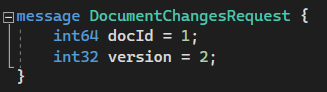
Для отправки изменений на сервер создается запрос, в котором требуется указать id-документа, версию и список изменений, содержащий тип изменения, позицию и изменяемый текст. Данная структура представлена на рисунке 7.



Рисунок

7 – Структура изменений документа

Для получения изменений документа, отправляется запрос на рабочий сервер, в котором необходимо указать id-документа и последнюю версию на клиенте (рис. 8). Ответом на такой запрос приходит структура, которая изображена на рисунке 7.



Рисунок

8 – Структура запроса на получение изменений документа

1. Отказоустойчивость системы

При отказе диспетчера система не сможет предоставить клиентам возможность получить список доступных документов и редактировать их, однако позволит клиентам изменять текущие открытые документы. Если диспетчер будет восстановлен на другом узле сети, то система возобновит свою полную работоспособность.

При отказе одного из нескольких рабочих серверов система будет успешно полностью функционировать. При этом для открытых документов клиент запросит у диспетчера другой из работающих серверов, если текущий сервер перестанет отвечать на запросы. При восстановлении рабочего сервера на другом узле сети, сам сервер уведомит диспетчера о своем появлении с указанием ему своего IP-адреса, таким образом пополнив пул доступных рабочих серверов.

При потере соединения с одним из серверов, клиент выдаст предупреждение о недоступности соответствующего сервера.

1. Тестирование с помощью Chaos Framework

Для тестирования распределенного приложения был использован Chaos Framework (<https://github.com/iskorotkov/chaos-framework>). Тестирование показало, что отказоустойчивость системы на высоком уровне: пользователи приложения не теряли способность редактировать документы, однако иногда могли наблюдать небольшие задержки синхронизации документов с сервером.

1. Исходный код приложения
   1. Клиент

***MainWindow.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Doc;

namespace Client

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private long docId;

private CommunicateServer server;

private WordProcessing wordProcessing;

private DocumentContentResponse documentBackup;

private List<DocumentsResponse.Types.DocumentInfo> documents;

private Timer timer;

private string success = "Успешное подключение к серверам";

public MainWindow()

{

InitializeComponent();

wordProcessing = new WordProcessing();

timer = new Timer(SyncDoc);

try

{

server = new CommunicateServer("trrp-disp.mooo.com");

server.ConnectToDispatcher();

documents = server.GetDocuments();

foreach (var doc in documents)

{

documentsList.Items.Add(doc.Title);

}

}

catch(Exception ex)

{

logTextBlock.Text = ex.Message;

}

}

private void GetActualDocument()

{

try

{

var docServer = server.GetDocServer(docId);

server.ConnectToDocumentServer(docServer.Address);

var actualDoc = server.GetActualDocumentContent(docId);

documentBackup = new DocumentContentResponse()

{

Text = actualDoc.Text,

Version = actualDoc.Version

};

ChangeTextInRichTextBox(actualDoc.Text);

logTextBlock.Text = success;

}

catch (UnavailableDispatcherServerException ex)

{

throw ex;

}

catch (UnavailableDocumentServerException ex)

{

throw ex;

}

}

private void LoadDocumentFromServer(object sender, RoutedEventArgs e)

{

try

{

var doc = documents.Where(z => z.Title == documentsList.SelectedItem.ToString()).First();

docId = doc.DocId;

server.ConnectToDispatcher();

GetActualDocument();

timer.Start();

}

catch (UnavailableDocumentServerException)

{

logTextBlock.Text = "Попытка переподключения к рабочему серверу провалилась";

}

catch (UnavailableDispatcherServerException)

{

logTextBlock.Text = "Попытка переподключения к диспетчеру провалилась";

}

}

private void SyncDoc(object sender, EventArgs e)

{

try

{

SendDiffToServer();

ReceiveDiffFromServer();

logTextBlock.Text = success;

}

catch (Exception ex)

{

logTextBlock.Text = ex.Message;

}

}

private void SendDiffToServer()

{

var newText = GetTextFromRichTextBox();

var oldText = documentBackup.Text;

var dataForSend = new DocumentChanges()

{

DocId = docId,

Version = documentBackup.Version

};

wordProcessing.Compare(oldText, newText, dataForSend);

try

{

if (dataForSend.Changes.Count != 0)

server.SendDocumentChanges(dataForSend);

}

catch(UnavailableDocumentServerException)

{

GetActualDocument();

}

}

private void ReceiveDiffFromServer()

{

DocumentChanges docChanges;

try

{

docChanges = server.GetDocumentChanges(docId, documentBackup.Version);

}

catch(NotFoundDocumentException)

{

GetActualDocument();

docChanges = server.GetDocumentChanges(docId, documentBackup.Version);

}

catch (UnavailableDocumentServerException)

{

GetActualDocument();

docChanges = server.GetDocumentChanges(docId, documentBackup.Version);

}

if (docChanges.Changes.Count != 0)

{

var text = documentBackup.Text;

var newText = wordProcessing.ApplyTextChanges(text, docChanges);

docId = docChanges.DocId;

documentBackup.Version = docChanges.Version;

documentBackup.Text = newText;

ChangeTextInRichTextBox(newText);

}

}

private void ChangeTextInRichTextBox(string text)

{

var docChanges = new DocumentChanges();

//wordProcessing.Compare(GetTextFromRichTextBox(), text, docChanges);

wordProcessing.Compare(documentBackup.Text, text, docChanges);

int prevCursorPos = GetCaretPos();

int offset = prevCursorPos;

for (int i = 0; i < docChanges.Changes.Count; i++)

{

var change = docChanges.Changes[i];

if (change.Pos >= prevCursorPos)

continue;

if (change.Type == DocumentChanges.Types.ChangeType.Insert)

{

offset += change.Text.Length;

}

else if (change.Type == DocumentChanges.Types.ChangeType.Delete)

{

offset -= change.Text.Length;

}

}

docBox.Document.Blocks.Clear();

docBox.Document.Blocks.Add(new Paragraph(new Run(text)));

docBox.CaretPosition = docBox.Document.ContentStart;

int nl\_count = 0;

for (int i = 0; i < text.Length; ++i)

{

if (i >= offset)

break;

if (text[i] == '\n')

nl\_count++;

}

for (int i = 0; i < Math.Abs(offset) - nl\_count && i < text.Length - nl\_count; ++i)

docBox.CaretPosition = docBox.CaretPosition.GetNextInsertionPosition(offset > 0 ? LogicalDirection.Forward : LogicalDirection.Backward);

}

private int GetCaretPos()

{

TextPointer start = docBox.Document.ContentStart;

TextPointer caret = docBox.CaretPosition;

TextRange range = new TextRange(start, caret);

return range.Text.Length;

}

private string GetTextFromRichTextBox()

{

var text = new TextRange(docBox.Document.ContentStart, docBox.Document.ContentEnd).Text;

return text.Substring(0, text.Length - 2);

}

}

}

***CommunicateServer.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using Doc;

using Grpc.Core;

namespace Client

{

class CommunicateServer

{

private string domain;

private Channel documentServerChannel;

private Channel dispatcherChannel;

private DocumentService.DocumentServiceClient documentClient;

private DispatcherService.DispatcherServiceClient dispatcherClient;

public CommunicateServer(string domain = "localhost")

{

//var ips = Dns.GetHostAddresses("trrp.mooo.com");

//var ip = "localhost";

//ip = ips[0].ToString();

this.domain = domain;

}

public void ConnectToDispatcher()

{

try

{

dispatcherChannel = new Channel(domain, 30163, ChannelCredentials.Insecure);

dispatcherClient = new DispatcherService.DispatcherServiceClient(dispatcherChannel);

}

catch (RpcException)

{

throw new Exception("Не удалось связаться с сервером диспечера");

}

}

public List<DocumentsResponse.Types.DocumentInfo> GetDocuments()

{

List<DocumentsResponse.Types.DocumentInfo> documents;

try

{

var request = new DocumentsRequest();

documents = dispatcherClient.GetDocuments(request, deadline: DateTime.UtcNow.AddSeconds(5)).Documents.ToList();

}

catch (RpcException)

{

throw new UnavailableDispatcherServerException("Сервер документов недоступен. Поторите попытку позже.");

}

return documents;

}

public DocServer GetDocServer(long docId)

{

DocServer serverResponse;

try

{

var serverRequest = new DocServerRequest() { DocId = docId };

serverResponse = dispatcherClient.GetDocServer(serverRequest, deadline: DateTime.UtcNow.AddSeconds(5));

}

catch (RpcException ex)

{

ConnectToDispatcher();

throw new UnavailableDispatcherServerException("Ошибка получения адреса сервера документа. Проблемы с доступом к диспетчеру.");

}

return serverResponse;

}

public void ConnectToDocumentServer(string host)

{

try

{

documentServerChannel = new Channel(host, ChannelCredentials.Insecure);

documentClient = new DocumentService.DocumentServiceClient(documentServerChannel);

}

catch(RpcException)

{

throw new UnavailableDocumentServerException("Рабочий сервер недоступен");

}

}

public DocumentContentResponse GetActualDocumentContent(long docId)

{

DocumentContentResponse document;

try

{

var request = new DocumentContentRequest() { DocId = docId };

document = documentClient.GetActualDocumentContent(request, deadline: DateTime.UtcNow.AddSeconds(5));

}

catch (RpcException ex)

{

throw new UnavailableDocumentServerException("Ошибка получения актуальной версии документа");

}

return document;

}

public DocumentChanges GetDocumentChanges(long docId, int version)

{

DocumentChanges documentChanges;

try

{

var request = new DocumentChangesRequest() {

DocId = docId,

Version = version

};

documentChanges = documentClient.GetDocumentChanges(request, deadline: DateTime.UtcNow.AddSeconds(5));

}

catch (RpcException e)

{

if (StatusCode.NotFound == e.StatusCode)

{

throw new NotFoundDocumentException();

}

else if (StatusCode.Unavailable == e.StatusCode)

{

throw new UnavailableDocumentServerException("Сервер документа недоступен. Проверьте соединение с сетью.");

}

else throw new UnavailableDocumentServerException("Ошибка при получении изменений документа.");

}

return documentChanges;

}

public void SendDocumentChanges(DocumentChanges documentChanges)

{

try

{

documentClient.SendDocumentChanges(documentChanges, deadline: DateTime.UtcNow.AddSeconds(5));

}

catch (RpcException e)

{

if (StatusCode.Unavailable == e.StatusCode)

{

throw new UnavailableDocumentServerException("Сервер документа недоступен. Проверьте соединение с сетью.");

}

else

throw new UnavailableDocumentServerException("Ошибка отправки изменений. Проверьте соединение с сетью.");

}

}

}

}

***WordProcessing.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DiffMatchPatch;

using Doc;

namespace Client

{

class WordProcessing

{

private diff\_match\_patch dmp;

public WordProcessing()

{

dmp = new diff\_match\_patch();

}

public void Compare(string oldText, string newText, DocumentChanges changes)

{

var diff = dmp.diff\_main(oldText, newText);

dmp.diff\_cleanupSemantic(diff);

GenerateDifferences(diff, changes);

}

private void GenerateDifferences(List<Diff> diffs, DocumentChanges changes)

{

long position = 0;

foreach (var diff in diffs)

{

switch(diff.operation)

{

case Operation.INSERT:

changes.Changes.Add(new DocumentChanges.Types.Change()

{

Type = DocumentChanges.Types.ChangeType.Insert,

Pos = position,

Text = diff.text

});

position += diff.text.Length;

break;

case Operation.DELETE:

changes.Changes.Add(new DocumentChanges.Types.Change()

{

Type = DocumentChanges.Types.ChangeType.Delete,

Pos = position,

Text = diff.text

});

break;

case Operation.EQUAL:

position += diff.text.Length;

break;

}

}

}

public string ApplyTextChanges(string text, DocumentChanges changes)

{

var sb = new StringBuilder(text);

foreach (var change in changes.Changes)

{

switch (change.Type)

{

case DocumentChanges.Types.ChangeType.Insert:

sb.Insert((int)change.Pos, change.Text);

break;

case DocumentChanges.Types.ChangeType.Delete:

sb.Remove((int)change.Pos, change.Text.Length);

break;

}

}

return sb.ToString();

}

}

}

* 1. Сервер-диспетчер

class DispatcherService(

dispatcher\_pb2\_grpc.DispatcherServiceServicer):

"""сервис по работе со множеством серверов

"""

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.doc\_servers = [

'trrp-disp.mooo.com:30001',

'trrp-disp.mooo.com:30002',

]

storage\_client = storage.Client()

self.doc\_bucket = storage\_client.get\_bucket(CFG['storage\_bucket'])

def GetDocServer(self, request, context):

"""получение сервера по ид документа

"""

# select random server

server\_idx = request.docId % len(self.doc\_servers)

server\_addr = self.doc\_servers[server\_idx]

# check server availability

channel = grpc.insecure\_channel(server\_addr)

stub = document\_pb2\_grpc.DocumentServiceStub(channel)

try:

stub.AddDocument(AddDocumentRequest(docId=request.docId, secretKey=CFG['secret\_key']))

except grpc.RpcError as e:

context.set\_code(grpc.StatusCode.UNAVAILABLE)

context.set\_details('Document server unavailable!')

return DocServer()

return DocServer(address=server\_addr)

def GetDocuments(self, request, context):

"""получение списка документов

"""

response = DocumentsResponse()

#for title in self.doc\_bucket.list\_blobs():

# response.documents.append(

# DocumentsResponse.DocumentInfo(docId=title\_to\_id(title), title=title))

response.documents.append(

DocumentsResponse.DocumentInfo(docId=1, title=f'doc1.txt'))

response.documents.append(

DocumentsResponse.DocumentInfo(docId=2, title=f'doc2.txt'))

return response

* 1. Сервер-документа

class DocumentService(document\_pb2\_grpc.DocumentServiceServicer):

"""сервис по работе с одним документом

"""

def \_\_init\_\_(self):

super().\_\_init\_\_()

session = boto3.Session()

self.client = session.client(

service\_name='s3',

endpoint\_url='https://storage.yandexcloud.net',

region\_name='ru-central1',

aws\_access\_key\_id='SLmdUdOyVS0t4nqBZsyM',

aws\_secret\_access\_key='On-r-v864dSjRCmpAMS6VsAKZjo\_qyy\_MntNTX7q')

self.active\_documents = {}

def AddDocument(self, request, context):

"""проверка доступности сервера

"""

if request.secretKey != CFG['secret\_key']:

context.set\_code(grpc.StatusCode.PERMISSION\_DENIED)

context.set\_details('Permission denied!')

return AddDocumentResponse()

docId = request.docId

if docId not in self.active\_documents:

# load the document and make it active

doc\_title = f"doc{docId}.txt"

get\_object\_response = self.client.get\_object(Bucket='trrp-doc-bucket', Key=doc\_title)

self.active\_documents[docId] = ds.DocumentSync(get\_object\_response['Body'].read().decode('utf-8'))

return AddDocumentResponse()

def GetActualDocumentContent(self, request, context):

"""получение контента документа актуальной версии

"""

doc\_sync = self.get\_doc\_sync(request.docId, context)

if not doc\_sync:

return DocumentContentResponse()

text, version = doc\_sync.get\_actual\_content()

return DocumentContentResponse(text=text, version=version)

def GetDocumentChanges(self, request, context):

"""получить изменения

"""

doc\_sync = self.get\_doc\_sync(request.docId, context)

if not doc\_sync:

return DocumentChanges()

if not self.version\_exists(doc\_sync, request.version, context):

return DocumentChanges()

doc\_changes = doc\_sync.get\_last\_changes(request.version)

return DocumentChanges(

docId = request.docId,

version=doc\_changes.version,

changes=[

DocumentChanges.Change(

type=ch.type,

pos=ch.pos,

text=ch.text) for ch in doc\_changes.changes]

)

def SendDocumentChanges(self, request, context):

"""отправить изменения

"""

doc\_sync = self.get\_doc\_sync(request.docId, context)

if not doc\_sync:

return DocumentChangesResponse()

if not self.version\_exists(doc\_sync, request.version, context):

return DocumentChangesResponse()

doc\_changes = ds.DocumentChanges(

changes=[ds.TextChange(ch.type, ch.pos, ch.text) for ch in request.changes],

version=request.version

)

doc\_sync.add\_changes(doc\_changes)

# save

text, version = doc\_sync.get\_actual\_content()

doc\_title = f"doc{request.docId}.txt"

self.client.put\_object(Bucket='trrp-doc-bucket',

Key=doc\_title,

Body=text)

return DocumentChangesResponse()

def get\_doc\_sync(self, doc\_id, context) -> ds.DocumentSync:

"""проверка наличия документа

"""

if doc\_id not in self.active\_documents:

context.set\_code(grpc.StatusCode.NOT\_FOUND)

context.set\_details('Document not found!')

return None

return self.active\_documents[doc\_id]

def version\_exists(self, doc\_sync: ds.DocumentSync, version: int, context):

"""проверка наличия версии

"""

if doc\_sync.version\_exists(version):

return True

context.set\_code(grpc.StatusCode.NOT\_FOUND)

context.set\_details("Version {} not found!".format(version))

return False