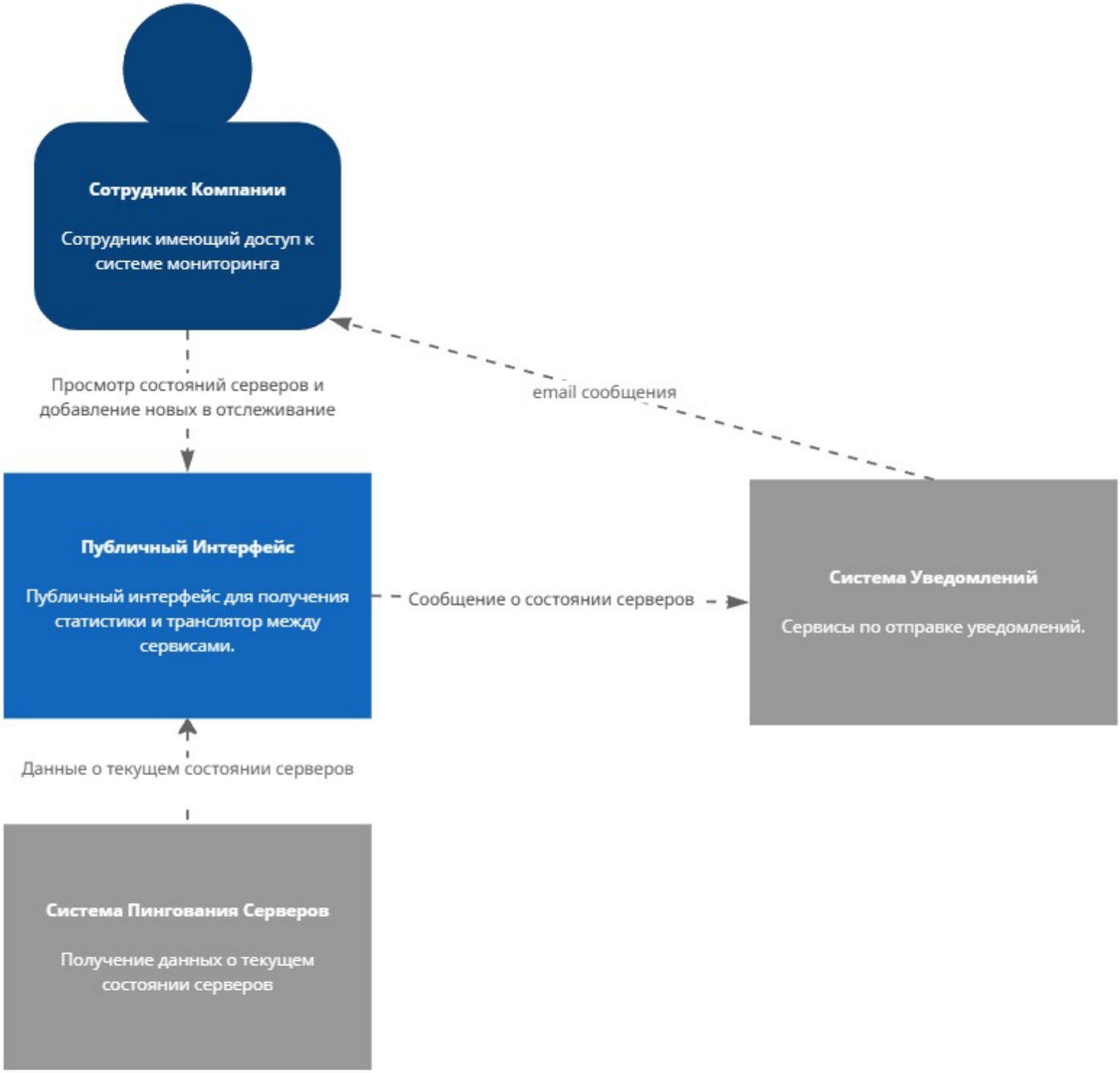


Диаграмма Контекста Системы для Системы Мониторинга

Общие сведения о Системе Мониторинга

Последнее обновление: 19.09.2025

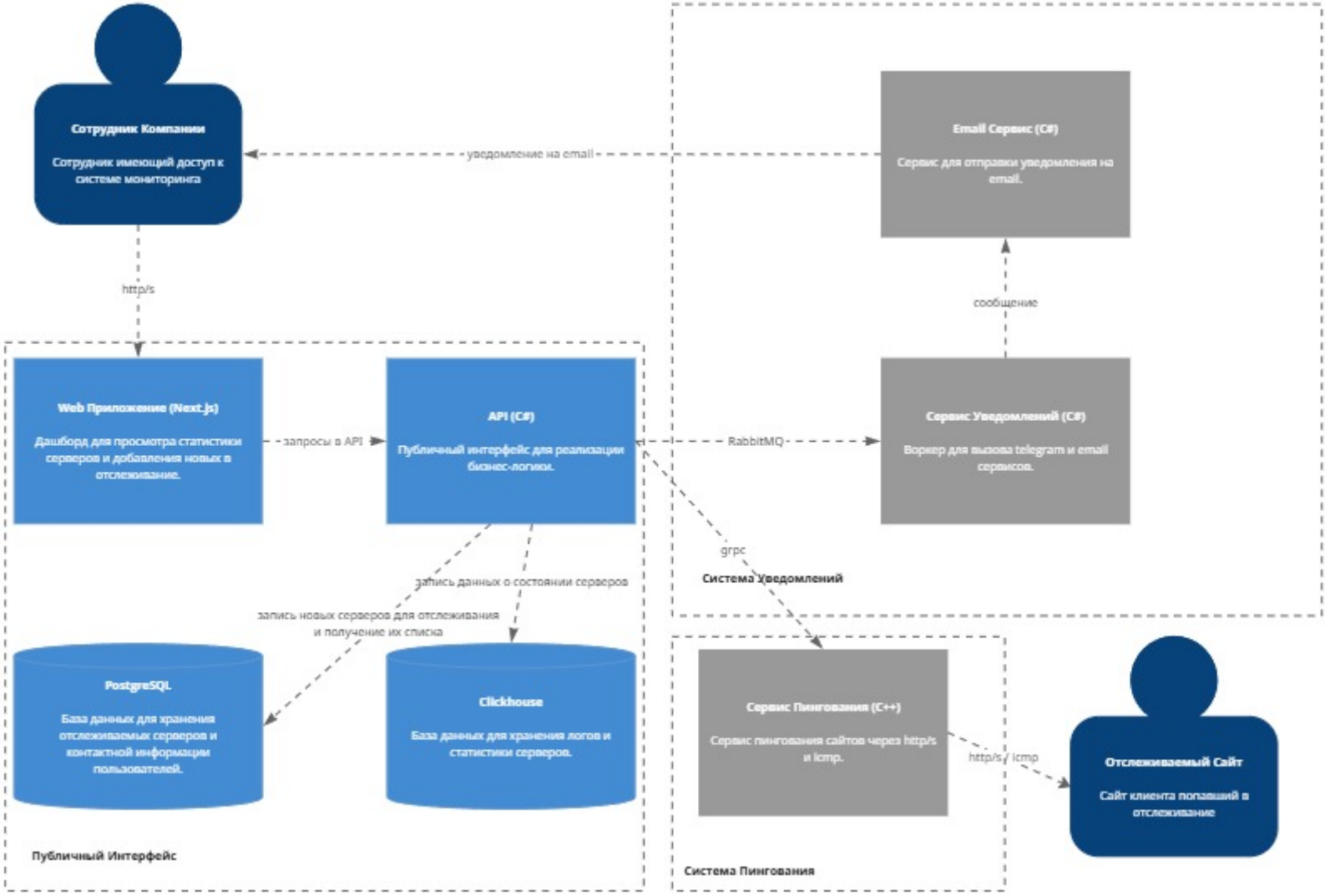


Диаграмма Контейнеров для Системы Мониторинга

Описание взаимодействия контейнеров Системы Мониторинга

Последнее обновление: 20.09.2025

	HTTP Method	Endpoint	Description	Request Parameters	Response Body

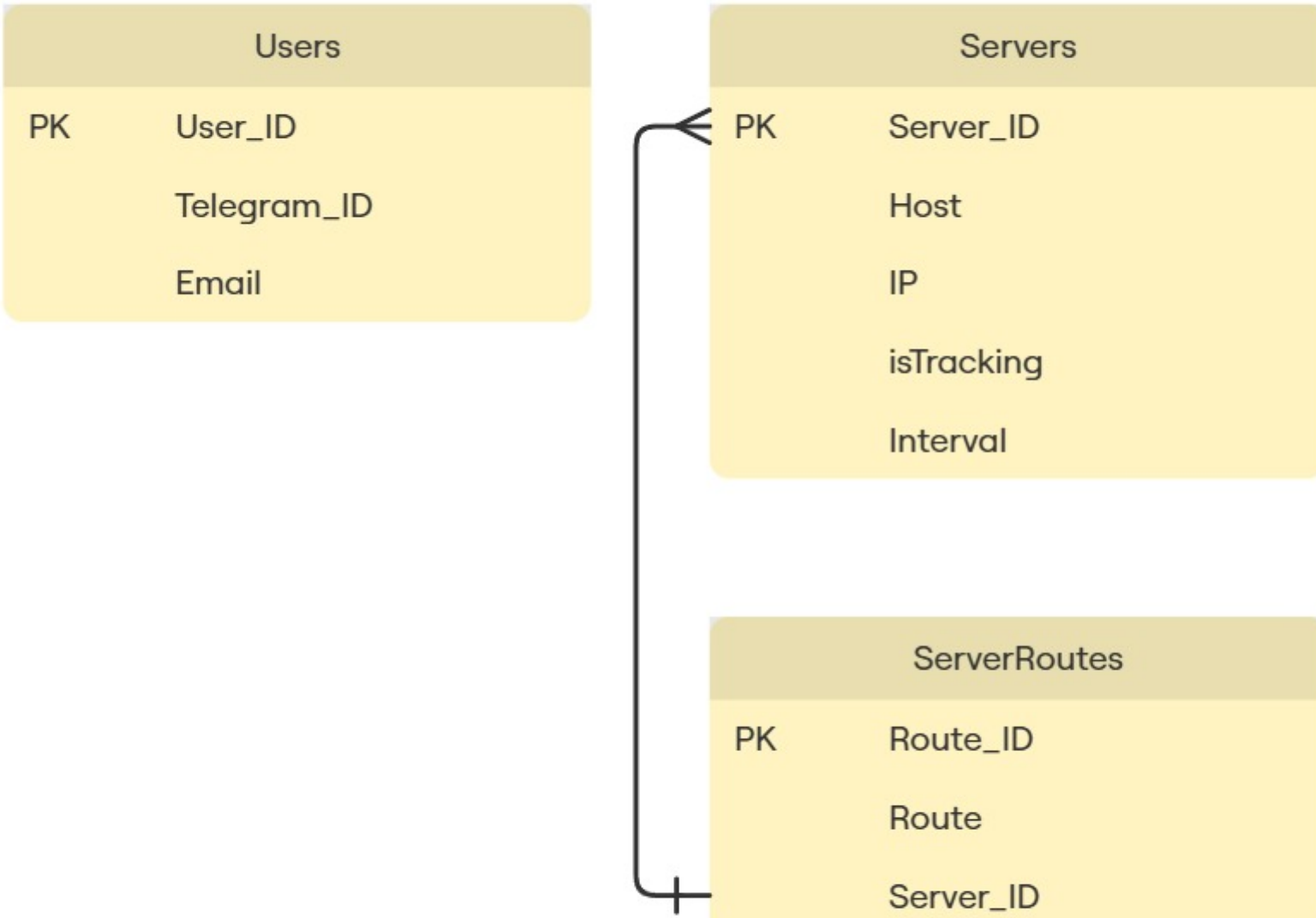
1	GET	/api/servers	Получение данных о состоянии серверов	Параметр запроса:
				query: string,
				limit: int = 25,
				offset: int,
					Массив json: [{
					id: int,
					host: string,
					ip: string,
					status: works | doesn'twork,
					protocol: HTTP | HTTPS | ICMP,
					++ errorMessage: string
					++ statusCode: string
					stats: {
					totalPings: int,
					successRate: double,
					avgResponseTimeMs: double
					lastCheck: string
					}
					}]

2	POST	/api/servers	Добавление нового сервера в отслеживание	Параметр body: {
				host: string,
				ip: string,
				interval: string,
				routes: [{
				route: string
				}]
				}
				
				h - hour
				m - minutes
					message: string

3	GET	/api/servers/{id}	Получение данных о состоянии сервера	Параметр маршрута:
				id: int
					json: {
					id: int,
					host: string,
					ip: string,
					status: works | doesn't work,
					protocol: HTTP | HTTPS | ICMP,
					++ errorMessage: string
					++ statusCode: string
					stats: {
					totalPings: int,
					successRate: double,
					avgResponseTimeMs: double
					lastCheck: string
					}
					}

4	PUT	/api/servers/{id}	Изменение параметров мониторинга	Параметр маршрута:
				id: int
				
				Параметр body: {
				interval: string
				}
				
				h - hour
				m - minutes
					message: string

5	DELETE	/api/servers/{id}	Удаление сервера из отслеживания	Параметр маршрута: id: int	message: string
6	GET	/api/servers/{id} /logs	Получение логов сервера	Параметр маршрута: id: int, Параметр запроса: limit: int = 50, offset: int = 0, from: datetime, to: datetime,	Массив json: [{ id: int, timestamp: string, responseTimeMs: double, success: bool, statusCode: int, protocol: HTTP | HTTPS | ICMP, errorMessage: string }]
7	GET	/api/servers/{id} stats	Получение данных для	Параметр запроса: { period: string = 24h,	Массив json: [{ id: int,


Users
PK
User_ID
Telegram_ID
Email
Servers
PK
Server_ID
Host
IP
isTracking
Interval
ServerRoutes
PK
Route_ID
Route
Server_ID

Описание

C# API:

у нас есть центральный компонент, это c# api. он общается с сервисом cpp через grpc и с сервисом уведомлений через rabbitmq. c# единственный имеет доступ к postgresql и clickhouse. postgresql хранит знания о серверах которые нужны для работы cpp и контактную информацию пользователей, которая нужна для отправки увед, а clickhouse хранит историю логов и сообщений с сервиса cpp. c# дожидается тайминга расписания и получается ответ от cpp, который записывается в clickhouse и отправляется в очередь в сервис уведомлений. запрос в rabbit из сервиса уведомлений поступает пустым (без данных), а c# отправляет в rabbit уже данные. данные для дашборда c# отправляет через конечные точки.

C++ Service:

сервис cpp занимается сбором информации о сайте. берет информацию о хостах из c# и опрашивает через заданный интервал, собирает статистику, проверяет ответы и отправляет это все на c#. общение происходит через grpc.

C# Sevice:

сервис уведомлений ждет сообщения из rabbitmq от c# и транслирует их пользователю.

Next.js Dashboard:

дашборд общается с c# посредством http запросов. он должен получать данные с конечных точек и рисовать из них страницы. на страницах должна быть цветовая индикация статуса работы, интерактивные графики доступности и временем отклика.