РЕФЕРАТ

Расчетно-пояснительная записка 0 с., 7 рис., 8 источн., 1 прил.

АУТЕНТИФИКАЦИЯ, АВТОРИЗАЦИЯ, МИКРОСЕРВИСЫ, ИНФРАСТРУКТУРНЫЕ СЕРВИСЫ, KUBERNETES, КЛАСТЕР, SIDECAR, OIDC, OAUTH2.0, REST API,

Цель работы: реализация программно-алгоритмического комплекса по авторизации инфраструктурных сервисов.

СОДЕРЖАНИЕ

РЕФЕРАТ **5**

ВВЕДЕНИЕ **9**

1 Аналитический раздел **11**
1.1 Микросервисная архитектура 11
1.2 Идентификация, аутентификация и авторизация 12
1.3 Аутентификация в микросервисах 13
1.4 Kubernetes и сайдкар контейнер 13
1.4.1 Kubernetes 13
1.4.2 Сайдкар контейнер 16
1.5 Протоколы аутентификации 16
1.5.1 OAuth 1.0 и OAuth 2.0 16
1.5.2 OpenID Connect 18
1.5.3 SAML 20

2 Конструкторский раздел **23**
2.1 Алгоритм аутентификации инфраструктурного сервиса 23
2.2 Алгоритм верификации k8s токена на стороне idP 24
2.3 Обработчики HTTP запросов к idP 25

3 Технологический раздел **26**
3.1 Развертывание k8s кластера 26
3.2 Реализация idP сервиса 32
3.3 Реализация клиента к idP 41
3.4 Тестирование программного обеспечения 49

4 Исследовательский раздел **52**
4.1 Описание проводимого исследования 52
4.2 Технические характеристики устройства 54
4.3 Полученные результаты 55

ЗАКЛЮЧЕНИЕ **56**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **58**

ПРИЛОЖЕНИЕ А Презентация **59**

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей расчетно-пояснительной записке к выпускной квалификационной работе применяют следующие сокращения и обозначения:

Sidecar	Паттерн сайдкар контейнера, при котором в одной сущности существует прокси-контейнер, расширяя возможности основного контейнера
k8s	Kubernetes — программное обеспечение для оркестрирования контейнеризированных приложений — автоматизации их развертывания, масштабирования и координации в условиях кластера
idP	identity provider — ключевая точка авторизации, запросы аутентификации проходят через нее, и в ней же выписываются OIDC токены

ВВЕДЕНИЕ

Последнее время все чаще можно услышать об утечке чувствительных персональных данных пользователей или компаний, занимающихся в таких сферах как FinTech, Healthcare, Enterprise. Отчасти это происходит из-за того, что компания и ее сотрудники небрежно следят за соблюдением информационной безопасности.

Информационная безопасность играет критически важную роль в распределенных системах с микросервисной архитектурой, поскольку такие системы часто состоят из множества независимых, взаимодействующих компонентов. Микросервисы обрабатывают и хранят большое количество данных, включая личную информацию пользователей и конфиденциальные бизнес-данные. Компрометация доступов к внутренним информационным системам и данным представляет собой финансовые, репутационные и правовые риски для компаний

В таких системах есть необходимость в эффективных и безопасных механизмах аутентификации и авторизации действий, доступных одному микросервису по отношению к другому, чтобы ограничить действия злоумышленника, получившего внутренний доступ. Особенно это касается инфраструктурных микросервисов — там, где доступ к данным совершается наиболее часто.

В данной работе будет реализована система авторизации инфраструктурных сервисов в системах с микросервисной архитектурой, чтобы исключить один из возможных этапов утечки чувствительных данных — несогласованный доступ как внутреннего сотрудника, так и злоумышленника извне систем. При этом работа авторизация не должна оказывать существенного влияния на работу системы, так как может быть внедрена в высоконагруженные системы.

Цель выпускной квалификационной работы — реализация программно-алгоритмического комплекса системы авторизации инфраструктурных сервисов.

Задачи выпускной квалификационной работы:

1) провести обзор существующих подходов аутентификации и авторизации в микросервисной архитектуре;

2) рассмотреть основные протоколы аутентификации, применимые в микросервисной архитектуре;

3) разработать и описать ключевые алгоритмы работы программно-алгоритмического комплекса авторизации инфраструктурных микросервисов;

4) разработать программное обеспечение, реализующее аутентификацию и авторизацию инфраструктурных микросервисов;

5) провести исследование влияния работы авторизации на выполнения запросов между инфраструктурными сервисами.

1 Аналитический раздел

1.1 Микросервисная архитектура

Микросервисы – это архитектурный и организационный подход к разработке программного обеспечения, при котором программное обеспечение состоит из небольших не зависимых сервисов, взаимодействующих через четко определенные интерфейсы API, обычно основанные на протоколах HTTP или gRPC. Эти сервисы принадлежат небольшим автономным командам. Архитектуры микросервисов упрощают масштабирование и ускоряют разработку приложений, позволяя внедрять инновации и ускоряя вывод новых функций на рынок.

В монолитных архитектурах все процессы тесно связаны и работают как единая служба. Это означает, что если один процесс приложения испытывает всплеск спроса, необходимо масштабировать всю архитектуру. Добавление или улучшение функций монолитного приложения усложняется по мере роста базы кода. Эта сложность ограничивает экспериментирование и затрудняет реализацию новых идей. Монолитные архитектуры повышают риск доступности приложений, поскольку множество зависимых и тесно связанных процессов увеличивают влияние сбоя одного процесса.

В архитектуре микросервисов приложение строится как независимые компоненты, которые запускают каждый процесс приложения как службу. Эти сервисы взаимодействуют через четко определенный интерфейс с использованием облегченных API. Службы созданы для бизнес-возможностей, и каждая служба выполняет одну функцию. Поскольку они запускаются независимо, каждую службу можно обновлять, развертывать и масштабировать в соответствии со спросом на определенные функции приложения. [1]

На рисунке изображено отличие в монолитном и микросервисных подходах 1.1.

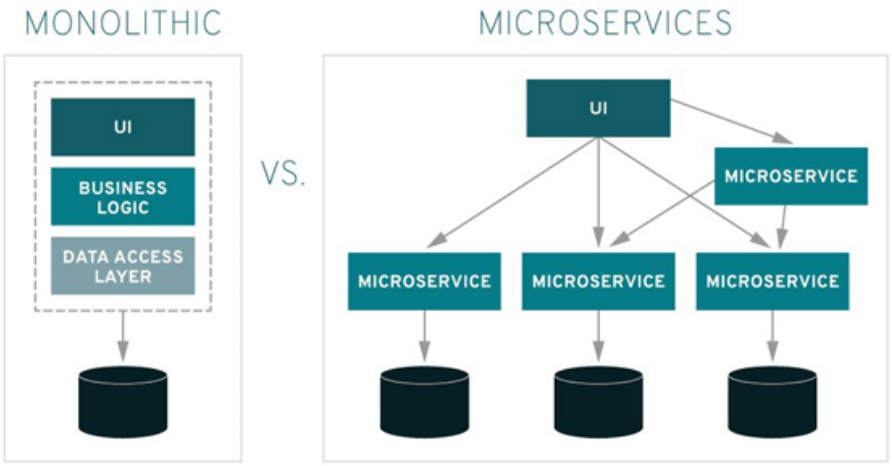

Рисунок 1.1 – Монолитная и микросервисная архитектура

Безопасность в микросервисных системах представляет собой важный аспект, обусловленный характером архитектуры, которая состоит из множества взаимодействующих компонентов. Каждый микросервис, функционирующий как автономная единица, взаимодействует с другими сервисами через API, что создает множество потенциальных точек уязвимости. Каждая точка входа является возможностью для несанкционированного доступа злоумышленников, стремящихся воспользоваться недостатками в системе. Компрометация одного микросервиса может иметь цепной эффект, влияя на другие компоненты системы. Кроме того, микросервисы часто обрабатывают конфиденциальную информацию, что обуславливает необходимость реализации шифрования как в состоянии покоя, так и в процессе передачи данных. [2]

1.2 Идентификация, аутентификация и авторизация

Идентификация представляет собой процесс, в ходе которого система определяет пользователя или объект, предоставляя уникальный идентификатор, такой как логин или номер учетной записи. Этот этап является первым шагом в установлении личности, позволяя системе распознать, с кем она взаимодействует, и подготовить соответствующий механизм контроля доступа.

Аутентификация представляет собой процесс проверки подлинности идентифицированного пользователя или объекта. На этом этапе система требует от пользователя подтверждения своей личности, что может осуществляться через различные методы, например ввод пароля. Прохождение аутентификация обеспечивает уверенность в том, что пользователь действительно является тем, за кого себя выдает.

Авторизация заключается в предоставлении пользователю системы прав доступа к определенным ресурсам или операциям на основании его роли, прав или политик безопасности. Этот процесс определяет, какие действия пользователь может выполнять и к каким ресурсам он имеет доступ, что позволяет минимизировать риски, связанные с несанкционированным доступом.

В рамках этих процессов важную роль играет концепция управления идентификацией и доступом (Identity and Access Management, IAM), которая включает в себя стратегии, технологии и практики, направленные на эффективное управление идентификацией пользователей и их правами доступа в организации. IAM обеспечивает централизованное управление учетными записями пользователей, автоматизацию процессов аутентификации и авторизации. [3]

1.3 Аутентификация в микросервисах

В условия микросервисной архитектуры, когда система состоит из множества независимых компонентов, каждый из которых может обрабатывать запросы от пользователей и других сервисов, есть необходимость в надежной аутентификации. В микросервисах аутентификация в основном реализуется через токены, такие как JSON Web Tokens (JWT), которые обеспечивают безопасный способ передачи информации, шифруя содержимое, о роли и правах одного микросервиса по отношению к другому.

Аутентификация также должна быть интегрирована с механизмами управления доступом, что позволяет не только удостовериться в личности пользователя, совершающего запрос, но и определить его права на доступ к определенным ресурсам и операциям.

Аутентификация в микросервисах должна быть масштабируема, справляясь с ростом количества микросервисов в системе, при этом не теряя в уровне безопасности. [4]

1.4 Kubernetes и сайдкар контейнер

1.4.1 Kubernetes

Работающий кластер Kubernetes включает в себя агента, запущенного на нодах (kubelet) и компоненты мастера (APIs, scheduler, etc), поверх решения

с распределённым хранилищем. Приведённая схема показывает желаемое, в конечном итоге, состояние, хотя все ещё ведётся работа над некоторыми вещами, например: как сделать так, чтобы kubelet (все компоненты, на самом деле) самостоятельно запускался в контейнере, что сделает планировщик подключаемым.

Узел (node) — это отдельная физическая или виртуальная машина, на которой развёрнуты и выполняются контейнеры приложений. Каждый узел в кластере содержит сервисы для запуска приложений в контейнерах (например Docker), а также компоненты, предназначенные для централизованного управления узлом.

Под (pod) — базовая единица для запуска и управления приложениями: один или несколько контейнеров, которым гарантирован запуск на одном узле, обеспечивается разделение ресурсов и межпроцессное взаимодействие и предоставляется уникальный в пределах кластера IP-адрес. Последнее позволяет приложениям, развёрнутым на поде, использовать фиксированные и предопределённые номера портов без риска конфликта. Поды могут напрямую управляться с использованием API Kubernetes или управление ими может быть передано контроллеру.

Том (volume) — общий ресурс хранения для совместного использования из контейнеров, развёрнутых в пределах одного пода.

Все объекты управления (узлы, поды, контейнеры) в Kubernetes помечаются метками (label), селекторы меток (label selector) — это запросы, которые позволяют получить ссылку на объекты, соответствующие какой-то из меток. Метки и селекторы — это главный механизм Kubernetes, который позволяет выбрать, какой из объектов следует использовать для запрашиваемой операции.

Сервисом в Kubernetes называют совокупность логически связанных наборов подов и политик доступа к ним. Например, сервис может соответствовать одному из уровней программного обеспечения, разработанного в соответствии с принципами многоуровневой архитектуры программного обеспечения. Набор подов, соответствующий сервису, получается в результате выполнения селектора соответствующей метки.

Kubernetes обеспечивает функции обнаружения сервисов и маршрутизации по запросу. В частности, система умеет переназначать необходимые

для обращения к сервису IP-адрес и доменное имя сервиса различным подам, входящим в его состав. При этом обеспечивается балансировка нагрузки в стиле Round robin DNS между подами, чьи метки соответствуют сервису, а также корректная работа в том случае, если один из узлов кластера вышел из строя и размещённые на нём поды автоматически были перемещены на другие узлы. По умолчанию сервис доступен внутри управляемого Kubernetes кластера — например, поды бэкенда группируются для обеспечения балансировки нагрузки и в таком виде предоставляются фронтенду. Также кластер может быть настроен и для предоставления доступа к входящим в его состав подам извне как к единому фронтенду.

Контроллер (controller) — это процесс, который управляет состоянием кластера, пытаясь привести его от фактического состояния к желаемому; он делает это, оперируя набором подов, определяемых с помощью селекторов меток и являющихся частью определения контроллера[19]. Выполнение контроллеров обеспечивается компонентом Kubernetes Controller Manager. Один из типов контроллеров, самый известный — это контроллер репликации (Replication Controller), который обеспечивает масштабирование, запуская указанное количество копий пода в кластере. Он также обеспечивает запуск новых экземпляров пода в том случае, если узел, на котором работает управляемый этим контроллером под, выходит из строя. Другие контроллеры, входящие в основную систему Kubernetes, включают в себя «DaemonSet Controller», который обеспечивает запуск пода на каждой машине (или подмножеством машин), и «Job Controller» для запуска подов, которые выполняются до завершения, например, как часть пакетного задания.

Операторы (operators) — специализированный вид программного обеспечения Kubernetes, предназначенный для включения в кластер сервисов, сохраняющих своё состояние между выполнениями (stateful), таких как СУБД, системы мониторинга или кэширования. Назначение операторов — предоставить возможность управления stateful-приложениями в кластере Kubernetes прозрачным способом и скрыть подробности их настроек от основного процесса управления кластером Kubernetes.

1.4.2 Сайдкар контейнер

Sidecar-контейнер — это контейнер, который должен быть запущен рядом с основным контейнером внутри пода. Этот паттерн нужен для расширения и улучшения функциональности основного приложения без внесения в него изменений. Sidecar-контейнеры могут использоваться для выполнения вспомогательных функций для main-контейнера: логирования, мониторинга, аутентификации, фоновых процессов или тестирования функциональности приложений.

Sidecar-контейнеры размещаются в той же внутренней сети, что и main-контейнер. Основное приложение может обращаться к sidecar-контейнеру, используя localhost с указанием порта, на котором работает sidecar-контейнер. Порты main- и sidecar-контейнера должны быть разными.

1.5 Протоколы аутентификации

В данном разделе будут рассмотрены протоколы аутентификации, структуры и сущности, которыми они оперируют, и их flow. Flow авторизации — это последовательность шагов, которые проходят объект авторизации, в данном случае микросервисы, для получения доступа к защищенным ресурсам. Все протоколы применяются как системах с пользователем и сервером, так и в распределенных системах с микросервисной архитектурой.

1.5.1 OAuth 1.0 и OAuth 2.0

OAuth (Open Authorization) — это открытый стандарт, разработанный для делегирования доступа к защищенным ресурсам без необходимости передачи учетных данных пользователя. Он стал основой для более совершенной версии — OAuth 2.0, выпущенной в 2012 году, описанный в стандарте RFC-6749 [5]. OAuth 2.0 расширяет возможности своего предшественника, предлагая более гибкую архитектуру и упрощенные механизмы авторизации, один из самых используемых в современных веб-приложениях и API. [6]

Протокол OAuth включает несколько ключевых сущностей:

1) ресурсный владелец (Resource Owner) — это пользователь, который предоставляет доступ к своим защищенным ресурсам;

2) клиент (Client) — приложение, которое запрашивает доступ к защищен-

ному ресурсу от имени resource owner и с его разрешения;

3) сервер авторизации (Authorization Server) — сервер, который отвечает за аутентификацию ресурсного владельца и выдачу токенов доступа. Он управляет процессом авторизации и хранит учетные данные пользователей;

4) ресурсный сервер (Resource Server) — это сервер, который хранит защищенные ресурсы и принимает запросы на доступ к ним на основании токенов доступа, выданных авторизационным сервером. Сервис может быть одновременно как клиентом, так и сервером.

OAuth 2.0 основывается на концепции токенов доступа, например JWT, которые используются для авторизации запросов к защищенным ресурсам. Основными компонентами структуры OAuth являются:

1) токен доступа (Access Token) — это строка с зашифрованным содержанием, которая представляет собой разрешение на доступ к защищенным ресурсам. Токены доступа имеют ограниченный срок действия и могут быть отозваны;

2) токен обновления (Refresh Token) — это токен, который используется для получения нового токена доступа после истечения его срока действия. Токены обновления позволяют клиенту продолжать доступ к ресурсам без повторной аутентификации пользователя;

3) клиентский идентификатор (Client ID) и секрет клиента (Client Secret) — эти идентификаторы используются для аутентификации клиента на авторизационном сервере.

OAuth 2.0 поддерживает несколько потоков авторизации (authorization flows), каждый из которых предназначен для определенных сценариев использования. Из них можно выделить основные:

1) Authorization code — этот flow основан на редиректах. Клиент должен уметь взаимодействовать с user-agent (обычно браузером) и обеспечивать клиент-серверное взаимодействие;

2) Client Credentials — этот flow больше остальных подходит для микросервисов. Авторизация выполняется на основе client id и client secret, что по сути из себя представляет логин и пароль, по которым сервер авторизации выдает клиенту access token к запрашиваемому ресурсу. На рисунке **??** представлен данный flow;

3) Implicit — по логике похож на Authorization code, но с тем отличием, что после успешной авторизации resource owner вместо получения authorization-code и его обмена на access token, сервер авторизации сразу возвращает access token в query в redirect URI. [5]

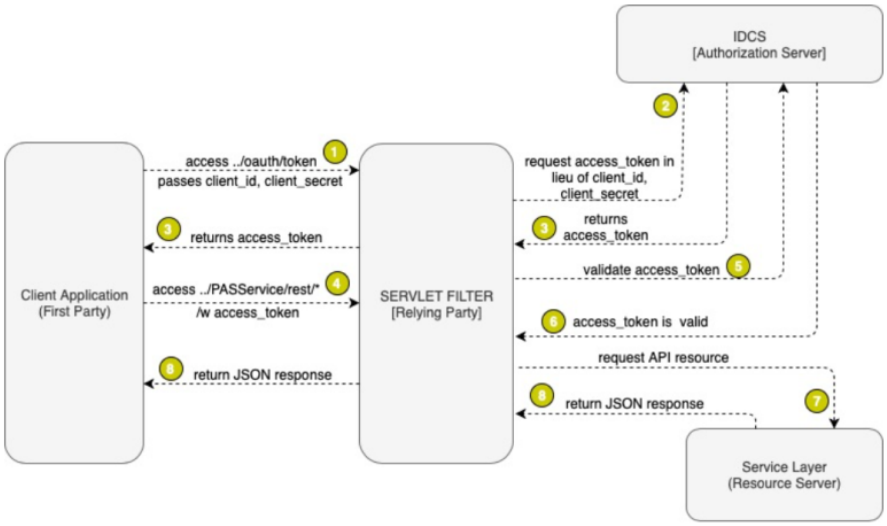

Рисунок 1.2 – OAuth2.0 Client Credentials flow авторизации

1.5.2 OpenID Connect

OpenID Connect (OIDC) представляет собой протокол аутентификации, построенный на основе OAuth 2.0, который позволяет клиентским приложениям проверять личность пользователей на основе аутентификации, выполненной авторизационным сервером. OAuth2.0 и OIDC часто используют вместе. [7]

Данный протокол содержит все те сущности и структуру, описанную в OAuth протоколе. Основные отличия данных протоколов:

1) ID Token — это токен, который содержит в зашифрованном виде информацию о пользователе, а также данные о сессии аутентификации. ID Token является основным элементом OpenID Connect и обычно кодируется в формате JWT (JSON Web Token). В OAuth2.0 токен не содержит информации о клиенте;

2) flow oidc позволяют клиентам получать как Access, так и ID Token;

3) в поле применения OIDC используется больше как протокол аутентификации, OAuth2.0 в связке с ним используется как протокол авторизации.

1.5.3 SAML

Протокол SAML (Security Assertion Markup Language) представляет собой стандарт, разработанный для обмена аутентификационной и авторизационной информацией между различными доменами безопасности.

Можно выделить основные сущности в данном протоколе:

1) Актор (Principal) — это пользователь или субъект, который пытается получить доступ к защищенным ресурсам. Актор инициирует процесс аутентификации;

2) Identity Provider (IdP) — это система, которая отвечает за аутентификацию акторов и выдачу утверждений (assertions). IdP проверяет личность пользователя и создает SAML-утверждения, которые содержат информацию о пользователе;

3) Service Provider (SP) — это система, которая предоставляет доступ к защищенным ресурсам. SP полагается на утверждения, выданные IdP, для принятия решения о том, предоставлять ли доступ пользователю;

4) SAML-утверждение (SAML Assertion) — это XML-документ, который содержит информацию о пользователе и его аутентификации. Утверждения могут содержать данные о том, как и когда пользователь был аутентифицирован, а также атрибуты, которые описывают пользователя.

Также можно выделить основную структуру протокола SAML:

1) SAML-Запрос (SAML Request) — сообщение, отправляемое SP в IdP с запросом на аутентификацию пользователя. Запрос может быть представлен в виде HTTP-запроса, содержащего параметры, такие как идентификатор запроса и URL-адрес, на который должно быть отправлено ответное сообщение;

2) SAML-Ответ (SAML Response) — сообщение, отправляемое IdP обратно в SP после успешной аутентификации пользователя. Ответ содержит SAML-утверждение, которое подтверждает аутентификацию и может содержать атрибуты пользователя;

3) XML — все сообщения SAML формализуются в виде XML-документов, что обеспечивает структурированный формат для обмена данными.

SAML имеет следующий flow авторизации, представленный на рисунке **??**:

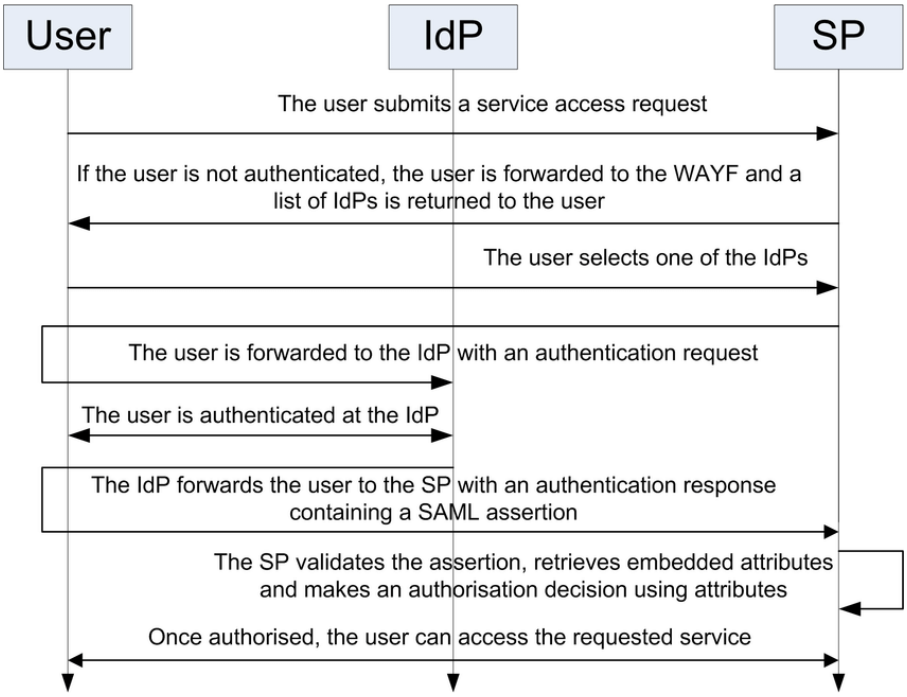

Рисунок 1.3 – SAML flow авторизации

1) инициация запроса на аутентификацию — актор пытается получить доступ к ресурсу на SP, который определяет, что актор не аутентифицирован, и перенаправляет его на IdP с SAML-запросом на аутентификацию. Запрос обычно кодируется в URL и передается через HTTP-редирект;

2) аутентификация пользователя — IdP получает SAML-запрос и инициирует процесс аутентификации. После успешной аутентификации IdP создает SAML-утверждение, которое содержит информацию о пользователе;

3) отправка SAML-ответа — IdP формирует SAML-ответ, который включает в себя SAML-утверждение, и перенаправляет актера обратно к SP. Ответ может быть передан через HTTP POST или HTTP Redirect;

4) обработка SAML-ответа — SP получает SAML-ответ и проверяет его подлинность. Если SAML-утверждение действительно, SP выполняет

авторизацию пользователя на основе атрибутов, содержащихся в утверждении (например, имя, адрес электронной почты, роли и права доступа);

5) доступ к ресурсу — после успешной авторизации SP предоставляет доступ к защищенным ресурсам актору. [8]

Вывод

В данном разделе были рассмотрены основные понятия микросервисной архитектуре, инструменте Kubernetes, паттерна Sidecar, а также протоколы аутентификации, применимые к микросервисной архитектуры и которые могут использоваться для аутентификации инфраструктурных сервисов.

2 Конструкторский раздел

2.1 Алгоритм аутентификации инфраструктурного сервиса

Для аутентификации сервиса-клиента, выполняющего подписанный запрос из сайдкар контейнера, необходим k8s токен пода, который хранится по пути с секретами. В обмен на этот k8s токен, idP сервис выпускает токен аутентификации, который будет использоваться для подписи запроса от имени сервиса-клиента. На рисунке 2.1 представлен алгоритм аутентификации сервиса-клиента.

Рисунок 2.1 – Алгоритм аутентификации инфраструктурного сервиса

2.2 Алгоритм верификации k8s токена на стороне idP

Для того, чтобы подтвердить подлинность k8s токена и личность, от имени которого он был выписан, idP сервису нужен получить JWKs сертификаты от Kubernetes API, с помощью них расшифровать JWT токен и затем его проверить. Алгоритм верификации k8s токена представлен на рисунке 2.2.

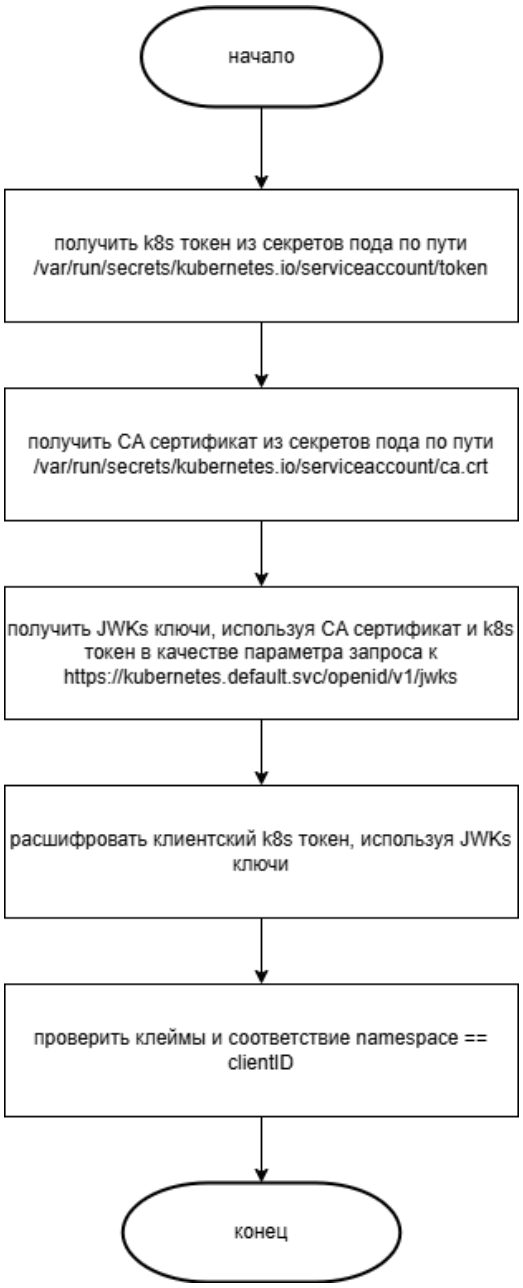

Рисунок 2.2 – Алгоритм верификации k8s токена

2.3 Обработчики HTTP запросов к idP

idP сервису необходимо реализовать REST API, чтобы в него могли ходить за выпуском токена. OIDC стандарт предлагает следующий контракт:

1) */realms/infra2infra/.well-known/openid-configuration* — обработчик запросов на получение OIDC конфигурации. Конфигурация должна содержать адреса и пути обработчиков запросов на выпуск токена и получения сертификатов, адрес issuer — где был выпущен токен, чтобы потом это проверить;

2) */realms/infra2infra/protocol/openid-connect/token* — обработчик запросов на выпуск и получение токена idP. Параметрами для запроса могут быть *"grant_type"* — способ проверки личности, в данном случае token-exchange, *"subject_token"* — сам токен для обмена, *"subject_token_type"* — тип токена для обмена, *"scope"* — в какую сущность выписывается токен;

3) */realms/infra2infra/protocol/openid-connect/certs* — обработчик запросов на получение сертификатов idP, возвращает публичный ключ сертификатов вместе с key-id.

Вывод

В данном разделе были спроектированы основные алгоритмы, необходимые для работы аутентификации, а также контракт общения idP и сервисов клиента по REST API.

3 Технологический раздел

Основные средства реализации:

1) k3d — утилита для поднятия k8s кластера локально, использует docker,
2) kubectl — утилита для ручного просмотра логов и состояния k8s кластера,
3) docker — инструмент для контейнеризации приложений. Используется в реализации для создания sidecar контейнеров,
4) ghcr.io — используется для загрузки docker образов в k8s кластер,
5) Golang — язык программирования, в основном использующийся для написания приложений в микросервисной архитектуре.

На рисунке 3.1 приведена структура реализованного проекта.

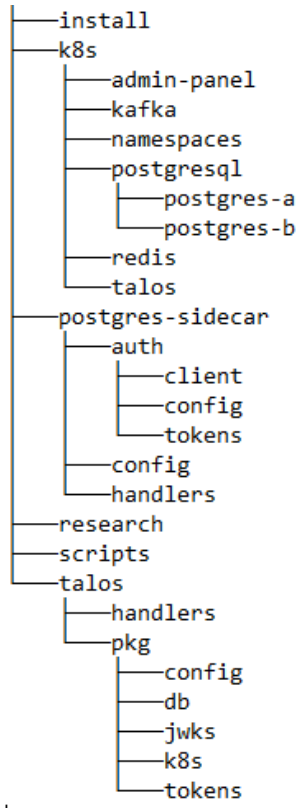

Рисунок 3.1 – Структура проекта

3.1 Развертывание k8s кластера

В листинге 3.1 приведен скрипт развертывания k8s сервисов.

Листинг 3.1 – Скрипт развертывания k8s кластера

```
#!/bin/bash

k3d cluster create bmstucluster \
  --api-port 6443 \
  --servers-memory 4G \
  --agents-memory 4G \
  --k3s-arg
     "--kubelet-arg=eviction-hard=memory.available<500Mi@server:*"
     \
  --k3s-arg
     "--kubelet-arg=eviction-hard=memory.available<500Mi@agent:*"
     \
  --k3s-arg "--kubelet-arg=image-gc-high-threshold=90@server:*" \
  --k3s-arg "--kubelet-arg=image-gc-low-threshold=80@server:*" \
  --k3s-arg "--kubelet-arg=fail-swap-on=false@server:*" \
  --kubeconfig-update-default \
  --k3s-arg "--kube-apiserver-arg=service-account-jwks-uri= \
        https://kubernetes.default.svc/openid/v1/jwks@server:*" \
  --k3s-arg "--kube-apiserver-arg=service-account-issuer= \
        https://kubernetes.default.svc@server:*"

# talos
docker build -t ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
   ./talos
docker push ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
k3d image import ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
   -c bmstucluster --keep-tools

# run sidecar code in sidecar containter:
docker build -t
   ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:latest
   ./postgres-sidecar
docker push
   ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:latest
k3d image import
   ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:latest -c
   bmstucluster --keep-tools

kubectl apply -f k8s/namespaces/
# kubectl apply -k k8s/namespaces/
```

```
namespaces=("postgres-a" "postgres-b" "talos")
for ns in "${namespaces[@]}"; do
  if ! kubectl get secret ghcr-secret -n "$ns" >/dev/null 2>&1;
     then
    kubectl create secret docker-registry ghcr-secret \
      --docker-server=ghcr.io \
      --docker-username=perpetua1g0d \
      --docker-password="$GH_PAT" \
      --namespace="$ns"
    echo "Secret GHCR created in namespace: $ns"
  else
    echo "Secret already exists in namespace: $ns"
  fi
done

kubectl apply -f k8s/talos/
kubectl apply -f k8s/postgresql/postgres-a/
kubectl apply -f k8s/postgresql/postgres-b/
```

В листингах 3.2 и 3.3 приведен пример конфигурации сервиса вместе с сайдкаром в одном поде, а также конфигурация развертывания сервиса idP.

Листинг 3.2 – Конфигурация развертывания PostgreSQL сервиса с сайдкаром

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-a
  namespace: postgres-a
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-a
  template:
    metadata:
      labels:
        app: postgres-a
    spec:
      serviceAccountName: default
      imagePullSecrets:
        - name: ghcr-secret
```

```
containers:
  - name: postgres
    image: postgres:13-alpine
    env:
      - name: POSTGRES_INITDB_ARGS
        value: "--data-checksums"
      - name: POSTGRES_PASSWORD
        value: "password"
      - name: POSTGRES_USER
        value: "admin"
      - name: POSTGRES_DB
        value: "appdb"
      - name: POSTGRES_PORT
        value: "5434"
    ports:
      - containerPort: 5434
    volumeMounts:
      - name: postgresql-data
        mountPath: /var/lib/postgresql/data
      - name: config
        mountPath: /etc/postgresql/postgresql.conf
        subPath: postgresql.conf
      - name: init-script
        subPath: init.sql
        mountPath: /docker-entrypoint-initdb.d/init.sql
      - name: shared-env
        mountPath: /etc/postgres-env
    lifecycle:
      postStart:
        exec:
          command:
            - "/bin/sh"
            - "-c"
            - |
              echo $POSTGRES_USER >
                /etc/postgres-env/POSTGRES_USER
              echo $POSTGRES_PASSWORD >
                /etc/postgres-env/POSTGRES_PASSWORD
              echo $POSTGRES_DB >
                /etc/postgres-env/POSTGRES_DB
              echo $POSTGRES_HOST >
```

```
                  /etc/postgres-env/POSTGRES_HOST
                echo $POSTGRES_PORT >
                  /etc/postgres-env/POSTGRES_PORT
      resources:
        limits:
          memory: "256Mi"
          cpu: "250m"

    - name: sidecar
      image:
        ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:lates
      volumeMounts:
        - name: shared-env
          mountPath: /etc/postgres-env
      env:
        - name: SERVICE_NAME
          value: "postgres-a"
        - name: SIGN_AUTH_ENABLED
          value: "false"
        - name: VERIFY_AUTH_ENABLED
          value: "false"
        - name: INIT_TARGET_SERVICE
          value: "postgres-b"
        - name: RUN_BENCHMARKS_ON_INIT
          value: "true"

        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      ports:
        - containerPort: 8080
      resources:
        limits:
          memory: "1G"
          cpu: "5"

  volumes:
    - name: postgresql-data
      emptyDir: {}
    - name: config
```

```
          configMap:
            name: postgresql-config
        - name: init-script
          configMap:
            name: postgres-init-script
        - name: shared-env
          emptyDir: {}
```

Листинг 3.3 – Конфигурация развертывания сервиса idP

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: talos
  namespace: talos
spec:
  replicas: 1
  selector:
    matchLabels:
      app: talos
  template:
    metadata:
      labels:
        app: talos
    spec:
      serviceAccountName: default
      imagePullSecrets:
        - name: ghcr-secret
      containers:
        - name: talos
          image:
      ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
          ports:
            - containerPort: 8080
          resources:
            limits:
              memory: "128Mi"
              cpu: "100m"
```

3.2 Реализация idP сервиса

idP сервис был реализован на языке программирования Golang и получил k8s кластере имя *talos*.

В листинге 3.4 приведена SQL схема infra2infra и таблица для хранения прав.

Листинг 3.4 – Схема и таблица для хранения прав

```
CREATE SCHEMA IF NOT EXISTS infra2infra;

CREATE TABLE infra2infra."Permissions" (
    ClientName TEXT NOT NULL,
    ServerName TEXT NOT NULL,
    roles TEXT[] NOT NULL
);
```

В листингах 3.5–3.6 приведен способ генерации сертификата, основанного на rsa ключе, а также структура клеймов токена.

Листинг 3.5 – Генерация сертификата

```
type KeyPair struct {
    PrivateKey  *rsa.PrivateKey
    Certificate *x509.Certificate
    KeyID       string
}

func GenerateKeyPair() *KeyPair {
    privateKey, _ := rsa.GenerateKey(rand.Reader, 2048)

    now := time.Now()
    template := &x509.Certificate{
        SerialNumber:          big.NewInt(1),
        Subject:               pkix.Name{CommonName:
           "talos-oidc"},
        NotBefore:             now,
        NotAfter:              now.Add(24 * time.Hour * 365),
        BasicConstraintsValid: true,
        KeyUsage:              x509.KeyUsageDigitalSignature |
           x509.KeyUsageKeyEncipherment,
    }

    certDER, _ := x509.CreateCertificate(
```

```
        rand.Reader,
        template,
        template,
        privateKey.Public(),
        privateKey,
    )

    cert, _ := x509.ParseCertificate(certDER)

    return &KeyPair{
        PrivateKey:  privateKey,
        Certificate: cert,
        KeyID:       generateKeyID(),
    }
}

func generateKeyID() string {
    const defaultLength = 24

    buf := make([]byte, defaultLength)
    rand.Read(buf)
    return base64.RawURLEncoding.EncodeToString(buf)
}

func GenerateJWT(signer jose.Signer, claims tokens.Claims)
   (string, error) {
    payload, err := json.Marshal(claims)
    if err != nil {
        return "", err
    }

    signature, err := signer.Sign(payload)
    if err != nil {
        return "", err
    }

    return signature.CompactSerialize()
}

func getX5t(cert *x509.Certificate) string {
    h := sha1.Sum(cert.Raw)
```

```
    return base64.RawURLEncoding.EncodeToString(h[:])
}

func getX5tS256(cert *x509.Certificate) string {
    h := sha256.Sum256(cert.Raw)
    return base64.RawURLEncoding.EncodeToString(h[:])
}
```

Листинг 3.6 – Структура клеймов токена

```
type Claims struct {
    Exp      time.Time `json:"exp"`
    Iat      time.Time `json:"iat"`
    Iss      string    `json:"iss"`
    Sub      string    `json:"sub"`
    Aud      string    `json:"aud"`
    Scope    string    `json:"scope"`
    Roles    []string  `json:"roles"`
    ClientID string    `json:"clientID"`
}
```

В листингах 3.7–3.9 приведена реализация OIDC обработчиков HTTP запросов к сервису idP. Обработчики слушают HTTP запросы на получение сертификатов по следующим путям:

1) */realms/infra2infra/.well-known/openid-configuration* — обработчик запросов на получение OIDC конфигурации,

2) */realms/infra2infra/protocol/openid-connect/token* — обработчик запросов на выпуск и получение токена idP,

3) */realms/infra2infra/protocol/openid-connect/certs* — обработчик запросов на получение сертификатов idP.

Листинг 3.7 – Реализация обработчика запросов на сертификаты

```
func CertsHandler(keys *jwks.KeyPair) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        jwks := keys.JWKS()
        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(jwks)
    }
}
```

```
func (k *KeyPair) JWKS() jose.JSONWebKeySet {
    jwk := jose.JSONWebKey{
        Key:          k.PrivateKey.Public(),
        Certificates: []*x509.Certificate{k.Certificate},
        KeyID:        k.KeyID,
        Algorithm:    "RS256",
        Use:          "sig",
    }

    return jose.JSONWebKeySet{Keys: []jose.JSONWebKey{jwk}}
}
```

Листинг 3.8 – Реализация обработчика запросов на OIDC конфигурацию

```
func OpenIDConfigHandler(cfg *config.Config) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        tokenEndpointPath :=
           "/realms/infra2infra/protocol/openid-connect/token"
        certsEndpointPath :=
           "/realms/infra2infra/protocol/openid-connect/certs"
        response := map[string]interface{}{
            "issuer":                                cfg.Issuer,
            "token_endpoint":                        cfg.Issuer
               + tokenEndpointPath,
            "jwks_uri":                              cfg.Issuer
               + certsEndpointPath,
            "grant_types_supported":
               []string{grantTypeTokenExchange},
            "id_token_signing_alg_values_supported":
               []string{"RS256"},
        }

        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(response)
    }
}
```

Листинг 3.9 – Реализация обработчика запросов на выпуск токена

```
const (
    grantTypeTokenExchange =
       "urn:ietf:params:oauth:grant-type:token-exchange" // RFC
       8693
```

```
    k8sTokenType            =
       "urn:ietf:params:oauth:token-type:jwt:kubernetes"
)

type TokenRequest struct {
    GrantType        string `form:"grant_type"`
    SubjectTokenType string `form:"subject_token_type"`
    SubjectToken     string `form:"subject_token"`
    Scope            string `form:"scope"`
}

func NewTokenHandler(ctx context.Context, cfg *config.Config,
   keys *jwks.KeyPair) (http.HandlerFunc, error) {
    issuer, err := NewIssuer(cfg, keys)
    if err != nil {
        return nil, fmt.Errorf("failed to create issued: %w",
           err)
    }

    k8sVerifier, err := k8s.NewVerifier(ctx)
    if err != nil {
        return nil, fmt.Errorf("failed to create k8s verifier:
           %w", err)
    }

    return func(w http.ResponseWriter, r *http.Request) {
        if err := r.ParseForm(); err != nil {
            log.Printf("failed to parse form request params:
               %v", err)
            http.Error(w, `{"error":"invalid_request"}`,
               http.StatusBadRequest)
            return
        }

        log.Printf("Incoming request: Method=%s, URL=%s,
           Body=%s", r.Method, r.URL, r.Form)

        req := TokenRequest{
            GrantType:        r.FormValue("grant_type"),
            SubjectTokenType: r.FormValue("subject_token_type"),
            SubjectToken:     r.FormValue("subject_token"),
```

```
            Scope:              r.FormValue("scope"),
        }

        if req.GrantType != grantTypeTokenExchange {
            log.Printf("unexpected grant_type: %s",
               req.GrantType)
            http.Error(w, `{"error":"unsupported_grant_type"}`,
               http.StatusBadRequest)
            return
        } else if req.SubjectTokenType != k8sTokenType {
            log.Printf("unexpected subject_token_type: %s",
               req.GrantType)
            http.Error(w,
               `{"error":"unsupported_subject_token_type"}`,
               http.StatusBadRequest)
            return
        }

        clientID, _, err :=
           k8sVerifier.VerifyWithClient(req.SubjectToken)
        if err != nil {
            log.Printf("failed to verify k8s token: %v", err)
            http.Error(w, `{"error":"token_not_verified"}`,
               http.StatusBadRequest)
            return
        }

        issueResp, err := issuer.IssueToken(clientID, req.Scope)
        if err != nil {
            log.Printf("failed to issue talos token: %v", err)
            http.Error(w, `{"error":"access_denied"}`,
               http.StatusForbidden)
            return
        }

        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(issueResp)

        log.Printf("token issued, clientID: %s, scope: %s",
           clientID, req.Scope)
    }, nil
```

```
}
```

В листингах 3.10–3.11 приведена реализация получения публичного k8s ключа сертификата проверки подписи k8s токена для последующего обмена на новый токен, который будет выпущен уже сервисом idP.

Листинг 3.10 – Получения публичного k8s ключа сертификата

```
type JWKS struct {
    Keys []JWK `json:"keys"`
}

type JWK struct {
    Kty string `json:"kty"`
    Kid string `json:"kid"`
    Use string `json:"use"`
    Alg string `json:"alg"`
    N   string `json:"n"`
    E   string `json:"e"`
}

func getPublicKey() (*rsa.PublicKey, error) {
    k8sCertPath :=
       "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt"
    caCert, err := os.ReadFile(k8sCertPath)
    if err != nil {
        return nil, fmt.Errorf("error reading CA cert: %w", err)
    }

    caCertPool := x509.NewCertPool()
    caCertPool.AppendCertsFromPEM(caCert)

    client := &http.Client{
        Transport: &http.Transport{
            TLSClientConfig: &tls.Config{
                RootCAs: caCertPool,
            },
        },
    }

    k8sTokenPath :=
       "/var/run/secrets/kubernetes.io/serviceaccount/token"
    token, err := os.ReadFile(k8sTokenPath)
```

```
    if err != nil {
        return nil, fmt.Errorf("error reading token: %w", err)
    }

    req, err := http.NewRequest("GET",
       "https://kubernetes.default.svc/openid/v1/jwks", nil)
    if err != nil {
        return nil, fmt.Errorf("creating k8s jwks request: %w",
           err)
    }
    req.Header.Add("Authorization", "Bearer "+string(token))

    resp, err := client.Do(req)
    if err != nil {
        return nil, fmt.Errorf("JWKS request failed: %w", err)
    }
    defer resp.Body.Close()

    var jwks JWKS
    if err := json.NewDecoder(resp.Body).Decode(&jwks); err !=
       nil {
        return nil, fmt.Errorf("JWKS parse error: %w", err)
    }

    if len(jwks.Keys) == 0 {
        return nil, errors.New("no keys in JWKS")
    }

    key := jwks.Keys[0]
    return makeRSAPublicKey(key)
}

func makeRSAPublicKey(key JWK) (*rsa.PublicKey, error) {
    nBytes, err := base64.RawURLEncoding.DecodeString(key.N)
    if err != nil {
        return nil, fmt.Errorf("invalid modulus: %w", err)
    }

    eBytes, err := base64.RawURLEncoding.DecodeString(key.E)
    if err != nil {
        return nil, fmt.Errorf("invalid exponent: %w", err)
```

```
    }

    return &rsa.PublicKey{
        N: new(big.Int).SetBytes(nBytes),
        E: int(new(big.Int).SetBytes(eBytes).Int64()),
    }, nil
}
```

Листинг 3.11 – Проверка k8s токена

```
type Verifier struct {
    publicKey *rsa.PublicKey
}

func NewVerifier(_ context.Context) (*Verifier, error) {
    publicKey, err := getPublicKey()
    if err != nil {
        return nil, fmt.Errorf("failed to get k8s public key:
           %w", err)
    }

    return &Verifier{
        publicKey: publicKey,
    }, nil
}

func (v *Verifier) VerifyWithClient(k8sToken string) (string,
   jwt.Claims, error) {
    var claims privateClaims
    token, err := jwt.ParseWithClaims(k8sToken, &claims,
       func(token *jwt.Token) (interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodRSA); !ok {
            return nil, fmt.Errorf("unexpected method: %v",
               token.Header["alg"])
        }
        return v.publicKey, nil
    })
    if err != nil {
        return "", nil, fmt.Errorf("parsing jwt: %v", err)
    }

    if !token.Valid {
        return "", claims, fmt.Errorf("token cannot be converted
```

```
            to known one, which means it is invalid")
    }

    podName := claims.Kubernetes.Pod.Name
    namespace := claims.Kubernetes.Namespace

    if podName == "" || namespace == "" {
        return "", claims, fmt.Errorf("invalid k8s token claims
            (pod: %s, namespace: %s)", podName, namespace)
    } else if !strings.HasPrefix(podName+"-", namespace) {
        return "", claims, fmt.Errorf("pod name and namespace
            must both start with service name (pod: %s,
            namespace: %s)", podName, namespace)
    }

    return claims.Kubernetes.Namespace, claims, nil
}
```

3.3 Реализация клиента к idP

В листингах 3.12–3.13 приведена реализация получения публичного сертификата idP и фонового получения токена для проверки токена входящего запроса.

Листинг 3.12 – Фоновое получение idP токена

```
type TokenSource struct {
    cfg *config.Config

    scope  string
    token  atomic.Pointer[string]
    issuer *Issuer

    refreshCh chan struct{}
    closeCh   chan struct{}
}

type TokenSet struct {
    sync.RWMutex

    set map[string]*TokenSource
}
```

```
func NewTokenSet(ctx context.Context, cfg *config.Config, scopes
   []string) (*TokenSet, error) {
    set := &TokenSet{
        set: make(map[string]*TokenSource),
    }
    for _, scope := range scopes {
        ts, err := NewTokenSource(ctx, cfg, scope)
        if err != nil {
            return nil, fmt.Errorf("failed to create tokensource
               for %s scope: %w", scope, err)
        }

        set.set[scope] = ts
    }

    return set, nil
}

func (t *TokenSet) Token(scope string) (string, error) {
    ts, ok := t.set[scope]
    if !ok {
        return "", fmt.Errorf("no tokensource for provided
           scope: %s", scope)
    }

    token := ts.Token()
    if token == "" {
        return "", fmt.Errorf("token is empty for provided
           scope: %s, check logs", scope)
    }

    return token, nil
}

func NewTokenSource(ctx context.Context, cfg *config.Config,
   scope string) (*TokenSource, error) {
    issuer := NewIssuer(cfg)

    ts := &TokenSource{
        cfg:       cfg,
```

```
        scope:     scope,
        issuer:    issuer,
        token:     atomic.Pointer[string]{},
        refreshCh: make(chan struct{}),
        closeCh:   make(chan struct{}),
    }

    go ts.runScheduler(context.WithoutCancel(ctx))

    return ts, nil
}

func (ts *TokenSource) runScheduler(ctx context.Context) {
    planner := time.NewTimer(0)
    defer planner.Stop()

    for {
        select {
        case <-ts.closeCh:
            return
        case <-ts.refreshCh:
        case <-planner.C:
        }

        if !ts.cfg.SignEnabled {
            resetTimer(planner, 1*time.Hour)
            continue
        }

        delay := func() (delay time.Duration) {
            tokenResp, err := ts.issuer.IssueToken(ctx, ts.scope)
            if err != nil {
                log.Printf("failed to issue token to %s scope:
                   %v", ts.scope, err)
                return ts.cfg.ErrTokenBackoff
            }

            accessToken := tokenResp.AccessToken
            ts.token.Store(&accessToken)

            expiry := tokenResp.ExpiresIn
```

```
            newDelay := calcDelay(time.Until(expiry))
            log.Printf("New token to %s scope has been issued,
               expiry: %s, until_next: %s", ts.scope, expiry,
               newDelay)
            return newDelay
        }()

        resetTimer(planner, delay)
    }
}

func calcDelay(ttl time.Duration) time.Duration {
    return time.Duration(rand.Float32() * float32(ttl))
}

// resetTimer stops, drains and resets the timer.
func resetTimer(t *time.Timer, d time.Duration) {
    if !t.Stop() {
        select {
        case <-t.C:
        default:
        }
    }

    t.Reset(d)
}
```

Листинг 3.13 – Проверка idP токена

```
const talosIssuer = "http://talos.talos.svc.cluster.local"

type tokenClaims struct {
    Exp      time.Time `json:"exp"`
    Iat      time.Time `json:"iat"`
    Iss      string    `json:"iss"`
    Sub      string    `json:"sub"`
    Aud      string    `json:"aud"`
    Scope    string    `json:"scope"`
    Roles    []string  `json:"roles"`
    ClientID string    `json:"clientID"`
}

type Verifier struct {
```

```
    cfg *config.Config

    certs *jose.JSONWebKeySet
}

func (v *Verifier) verifyClaims(claims *tokenClaims, needRoles
   []string) error {
    if claims.Scope != claims.Aud || claims.Scope !=
       v.cfg.ClientID {
        return fmt.Errorf("scope or aud is unexpected, service:
           %s, scope: %s, aud: %s", v.cfg.ClientID,
           claims.Scope, claims.Aud)
    } else if claims.Iss != talosIssuer {
        return fmt.Errorf("unexpected issuer, expected: %s, got:
           %s", talosIssuer, claims.Iss)
    } else if expired := claims.Exp.Before(time.Now()); expired {
        return fmt.Errorf("token is expired, exp: %s, now: %s",
           claims.Exp, time.Now())
    } else if rolesOk := lo.Every(claims.Roles, needRoles);
       !rolesOk {
        return fmt.Errorf("roles mismatched, want: %v, got: %v",
           needRoles, claims.Roles)
    }

    return nil
}

func verifyToken(rawToken string, certs *jose.JSONWebKeySet)
   (*tokenClaims, error) {
    token, err := jwt.ParseSigned(rawToken)
    if err != nil {
        log.Printf("failed to parse token: %v", err)
        return nil, fmt.Errorf("failed to parse token: %w", err)
    }

    var claims tokenClaims
    for _, header := range token.Headers {
        keys := certs.Key(header.KeyID)
        if len(keys) == 0 {
            continue
        }
```

```
        for _, key := range keys {
            if err := token.Claims(key.Public(), &claims); err
               == nil {
                return &claims, nil
            }
        }
    }

    log.Printf("no certificate found to parse token. certs: %v,
       tokenHeaders: %v", certs, token.Headers)
    return nil, fmt.Errorf("no certificate found to parse token")
}

func (v *Verifier) fetchJWKs(ctx context.Context)
   (*jose.JSONWebKeySet, error) {
    talosCertEndpoint := v.cfg.CertsEndpointAddress
    req, err := http.NewRequestWithContext(ctx, http.MethodGet,
       talosCertEndpoint, nil)
    if err != nil {
        return nil, fmt.Errorf("failed to create talos certs
           request: %w", err)
    }
    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{Timeout: v.cfg.RequestTimeout}
    resp, err := client.Do(req)

    var respBytes []byte
    if resp != nil && resp.Body != nil {
        respBytes, _ = io.ReadAll(resp.Body)
    }
    if err != nil {
        log.Printf("failed to get talos certs: %v; respBody:
           %s", err, string(respBytes))
        return nil, fmt.Errorf("failed to get talos certs: %w",
           err)
    }
    defer resp.Body.Close()

    var jwks jose.JSONWebKeySet
```

```
    if marshalErr := json.Unmarshal(respBytes, &jwks);
       marshalErr != nil {
        log.Printf("failed to unmarshal certs: %v; body: %s",
           marshalErr, string(respBytes))
        return nil, fmt.Errorf("failed to unmarshal certs
           response: %w", err)
    }

    return &jwks, nil
}
```

Пример использования клиента к idP для проверки токена из входящего HTTP запроса в инфраструктурном сервисе приведен на листинге 3.14

Листинг 3.14 – Проверка токена входящего запроса

```
type QueryRequest struct {
    SQL    string `json:"sql"`
    Params []any  `json:"params"`
}

func NewQueryHandler(ctx context.Context, authClient
   *auth_client.AuthClient) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        log.Printf("Incoming request: %s %s", r.Method, r.URL)

        cfg := config.GetConfig()

        if cfg.VerifyAuthEnabled {
            token := r.Header.Get("X-I2I-Token")
            if token == "" {
                respondError(w, "missing token",
                   http.StatusUnauthorized)
                return
            }

            requiredRole := "RO"
            sqlQuery := strings.ToUpper(r.URL.Query().Get("sql")
            if !strings.Contains(sqlQuery), "SELECT") {
                requiredRole = "RW"
            }

            if verifyErr := authClient.VerifyToken(token,
```

```
            []string{requiredRole}); verifyErr != nil {
            log.Printf("failed to verify token: %v",
                verifyErr)
            respondError(w, "forbidden: token has no
                required roles", http.StatusUnauthorized)
            return
        }

        log.Printf("successfully verified incoming token")
    }

    db, err := sql.Open("postgres", fmt.Sprintf(
        "host=%s port=%s user=%s password=%s dbname=%s
            sslmode=disable",
        cfg.PostgresHost,
        cfg.PostgresPort,
        cfg.PostgresUser,
        cfg.PostgresPassword,
        cfg.PostgresDB,
    ))
    if err != nil {
        respondError(w, "database connection failed",
            http.StatusInternalServerError)
        return
    }
    defer db.Close()

    var req QueryRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err !=
       nil {
        respondError(w, fmt.Sprintf("invalid request: %v",
            err), http.StatusBadRequest)
        return
    }

    start := time.Now()
    rows, err := db.Query(req.SQL, req.Params...)
    if err != nil {
        respondError(w, fmt.Sprintf("query failed: %v",
            err), http.StatusBadRequest)
        return
```

```
        }
        defer rows.Close()

        json.NewEncoder(w).Encode(map[string]interface{}{
            "status":  "success",
            "latency": time.Since(start).String(),
        })
    }
}
```

3.4 Тестирование программного обеспечения

Для функционального тестирования были написаны unit-тесты. Пример unit теста, реализованного с использованием Arrange-Act-Assert паттерна, для проверки k8s токена приведен в листинге 3.15.

Листинг 3.15 – Тест для проверки k8s токена

```
func Test_Verify(t *testing.T) {
    // Arrange
    token := "eyJhbGciOiJSUzI1NiIsImtpZCI6Ilp2UFNpYlpISzk1YlhGRjh
        KSjJlY25MWHZRZ29aV25lMXB3UV9IYUh5TmsifQ.eyJhdWQiOlsiaHR0c
        HM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiLC
        JrM3MiXSwiZXhwIjoxNzc5NjUxMTU2LCJpYXQiOjE3NDgxMTUxNTYsIml
        zcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2YyIsImp0aSI6
        ImM0ZWUzNWFmLTFmZWYtNDNmYi1iNDYyLTJiNjk0OTM5MTdjMCIsImt1Y
        mVybmV0ZXMuaW8iOnsibmFtZXNwYWNlIjoicG9zdGdyZXMtYSIsIm5vZG
        UiOnsibmFtZSI6ImszZC1ibXN0dWNsdXN0ZXItc2VydmVyLTAiLCJ1aWQ
        iOiI3YzEwM2NlOC05OTA2LTQ3NWMtOGM5Ni1jNGZiZjIyOWM1YTAifSwi
        cG9kIjp7Im5hbWUiOiJwb3N0Z3Jlcy1hLTY3OTRmY2I1ZjctcWI5em0iL
        CJ1aWQiOiI5MTE3NDllNy01NTFiLTQ2M2UtYmNmZC03ZjEyNGFhZTgxNW
        UifSwic2VydmljZWFjY291bnQiOnsibmFtZSI6ImRlZmF1bHQiLCJ1aWQ
        iOiIyNDhkNDE3Mi0xZDdlLTRiMWEtYmRlNS05NzAyN2FiNDFlMmQifSwi
        d2FybmFmdGVyIjoxNzQ4MTE4NzYzfSwibmJmIjoxNzQ4MTE1MTU2LCJzd
        WIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6cG9zdGdyZXMtYTpkZWZhdW
        x0In0.spCWGajmAAENHgK5zG5NTX2dz82S0gunARu9-ncvurmV5XqEKOK
        SEypC9Rm3ap2WfwOei0zm4-0Hbi3fdYZUZkSDZO-HLSHxPZcQoGvVjiKD
        OAIRDGwNb8bNVqkPPyy1Q8K5cD1anJwPiXYFzIQ4ZKDu_Ikp5ajkA3KYp
        ZUmPLFQ3a09k8ycTpvdnjSzCfIdEaUiO4jFlGJWEYKGo9XuY0VVjyyjwJ
        AdDOj6Ry0aIJzJzQTjS-IUs_dL7XVQUSlxp4mZvhLhnrhiL6uU59tX1QV
        tliZ3MgkO3XN_F2G5kuoigFN0NQp8EpiCeC2-e9T-rchZ3MR8KbPp8lgs
        4SA1iA"
```

```
jwk := JWK{
    N: `xItwcttR4qVTD4bBfsUDgpICFnoBk1H8qyN3jSVemH1wlPyn6CLn2
    aUmjQHW25f2LcraZr1_t7l0ogmaR46Gn7uyYGBEtIsNnjvvoAUVbmd8vI
    hPJI9flzDjJys4CEjefo1YFooD4YfqDei0GEYG2TYy42mO3TR6O3--47P
    bLIyZ2cbmHTwU-t_apqc3NUs0Sd6_gjDb0hrX0cFlOvBfL-J-3XEe4Zxe
    w7-qDnjQGIKdSEgS-v-wCwr30iqK9yDfHO9cHUtRNirLb4dybeOh3_vBM
    MLCVpKtH6GonDEVyRv7qJCigEinpHB78Uq0PAb_l8SOHougk2qp8-Cp3n
    b7rw`,
    E: "AQAB",
}

wantAud := jwt.ClaimStrings{
    "https://kubernetes.default.svc.cluster.local",
    "k3s",
}

publicKey, err := makeRSAPublicKey(jwk)
if err != nil {
    t.Fatalf("failed to create public rsa key: %v", err)
}

verifier := &Verifier{
    publicKey: publicKey,
}

// Act
gotClientID, gotClaims, gotErr :=
   verifier.VerifyWithClient(token)

// Assert
if gotErr != nil {
    t.Errorf("failed to verify token: %v", gotErr)
} else if gotClientID != testClientID {
    t.Errorf("expected clientID: %s, got: %s", testClientID,
       gotClientID)
}

gotAud, gotAudErr := gotClaims.GetAudience()
if gotAudErr != nil {
    t.Errorf("got unexpected aud err: %v", gotAudErr)
```

```
    }
    assert.EqualValues(t, wantAud, gotAud)
}
```

Вывод

В данном разделе были описаны средства реализации программного-алгоритмического комплекса, способы развертывания k8s кластера, в том числе сервисов с сайдкаром, приведена реализация idP сервиса, а также пример тестирования сервиса idP.

4 Исследовательский раздел

В исследовании будет проведено сравнение времени выполнения запроса с включенной и выключенной авторизацией от одного инфраструктурного сервиса к другому.

4.1 Описание проводимого исследования

На листингах 4.1–4.2 приведена реализация запроса к инфраструктурному сервису, а также запуск выполнения фиксированного количества запросов параллельно.

Листинг 4.1 – Реализация запроса к PostgreSQL сервису

```
func sendBenchmarkQuery(cfg *config.Config, authClient
   *auth_client.AuthClient) {
    target :=
       fmt.Sprintf("http://%s.%s.svc.cluster.local:8080%s",
        cfg.InitTarget,
        cfg.InitTarget,
        cfg.ServiceEndpoint,
    )

    reqBody, _ := json.Marshal(map[string]interface{}{
        "sql":    `INSERT INTO log (message) VALUES ($1)`,
        "params": []interface{}{fmt.Sprintf("Write from %s, ts:
           %s", cfg.Namespace, time.Now())},
    })

    req, err := http.NewRequest("POST", target,
       bytes.NewBuffer(reqBody))
    if err != nil {
        log.Fatalf("failed to create post request: %v", err)
        return
    }

    if cfg.SignAuthEnabled {
        token, err := authClient.Token(cfg.InitTarget)
        if err != nil {
            log.Fatalf("failed to issue token in auth client on
               scope %s: %v", cfg.InitTarget, err)
            return
        }
```

```
        req.Header.Set("X-I2I-Token", token)
    }
    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{Timeout: 4 * time.Second}
    resp, err := client.Do(req)

    errMsg := handlers.RespErr{}
    var respBytes []byte
    if resp != nil && resp.Body != nil {
        respBytes, _ = io.ReadAll(resp.Body)
        _ = json.Unmarshal(respBytes, &errMsg)
    }

    if err != nil {
        log.Fatalf("Initial query failed: %v; errMsg: %s", err,
           errMsg.Error)
        return
    }
    defer resp.Body.Close()
}
```

Листинг 4.2 – Реализация запуска выполнения параллельных запросов

```
func runBenchmarks(cfg *config.Config, authClient
   *auth_client.AuthClient) {
    file, err := os.Create(benchmarksResultsFile)
    if err != nil {
        log.Fatalf("Cannot create results file: %v", err)
    }
    defer file.Close()
    writer := csv.NewWriter(file)
    defer writer.Flush()
    writer.Write([]string{"requests", "time_ms", "operation",
       "sign_enabled", "sign_disabled"})

    requestCount := []int64{100, 250, 500, 750, 1000}
    rerunCount := 10
    for _, reqCount := range requestCount {
        var avgTime float64 = 0
        for _ = range rerunCount {
            wg := &sync.WaitGroup{}
            wg.Add(int(reqCount))
```

```
            start := time.Now()
            for i := 0; i < int(reqCount); i++ {
                go func() {
                    defer wg.Done()
                    sendBenchmarkQuery(cfg, authClient)
                }()
            }
            wg.Wait()

            duration := time.Since(start).Milliseconds()
            avgTime += float64(duration)
        }

        avgTime = avgTime / float64(rerunCount*int(reqCount))
        log.Printf("finished %d requests, avg: %f", reqCount,
           avgTime)
        writer.Write([]string{
            strconv.FormatInt(reqCount, 10),
            strconv.FormatFloat(avgTime, 'f', 2, 64),
            "write",
            fmt.Sprintf("%v", cfg.SignAuthEnabled),
            fmt.Sprintf("%v", cfg.VerifyAuthEnabled),
        })
    }
}
```

4.2 Технические характеристики устройства

Технические характеристики устройства, на котором проводилось исследование:

1) процессор Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz,

2) оперативная память 8 ГБ,

3) операционная система Ubuntu 21.0.

Исследование проводилось на ноутбуке. Во время исследования ноутбук не был нагружен посторонними приложениями, которые не относятся к исследованию, а также ноутбук был подключен к сети питания.

4.3 Полученные результаты

Исследование проводилось при включенной и выключенной авторизации 100, 250, 500, 750, 1000 параллельных запросов из одного инфраструктурного сервиса к другому. Результаты для каждого количества запросов были усреднены путем запуска 10 раз.

Графики полученных усредненных результатов представлены на рисунке 4.1.

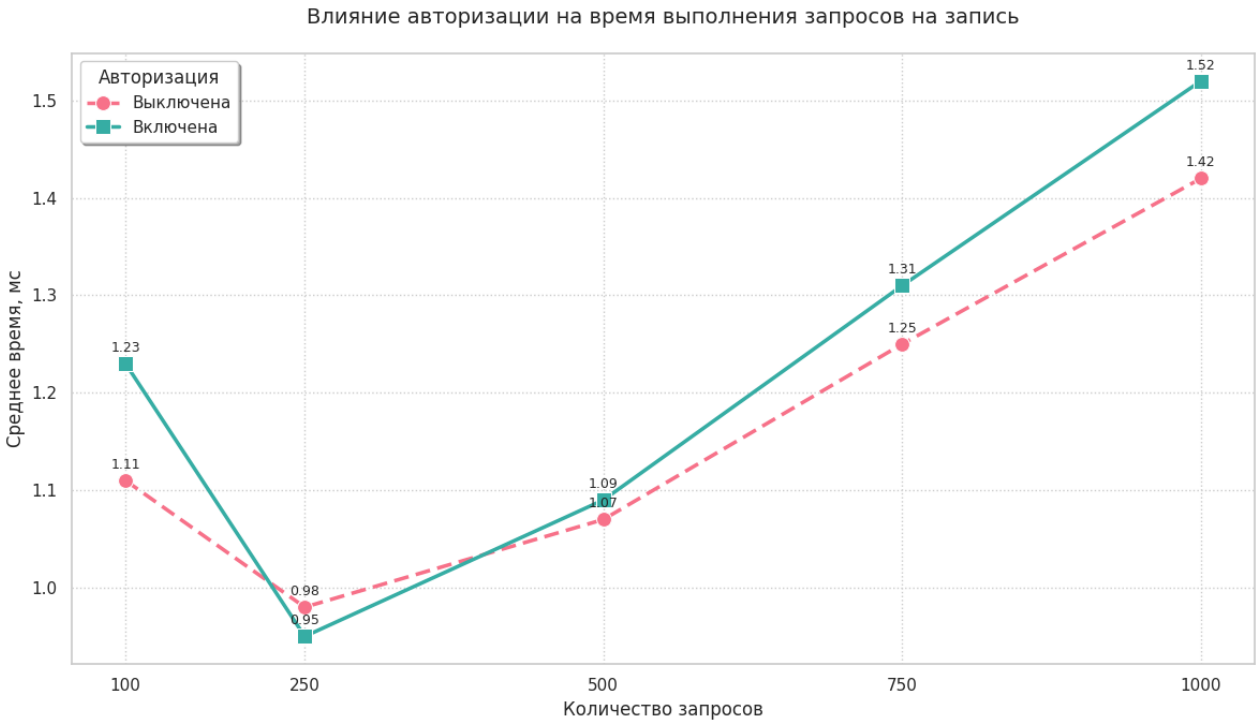

Рисунок 4.1 – Результаты исследования

Вывод

Как видно по графикам, время выполнения запросов с включенной авторизацией оказались в среднем на 10% дольше времени выполнения запросов с выключенной авторизацией. Это не окажет существенного влияния на работу системы из инфраструктурных сервисов.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы был разработан программно-алгоритмический комплекс авторизации инфраструктурных сервисов.

В ходе выполнения данной работы были решены следующие задачи:

1) провести обзор существующих подходов аутентификации и авторизации в микросервисной архитектуре;

2) рассмотреть основные протоколы аутентификации, применимые в микросервисной архитектуре;

3) разработать и описать ключевые алгоритмы работы программно-алгоритмического комплекса авторизации инфраструктурных микросервисов;

4) разработать программное обеспечение, реализующее аутентификацию и авторизацию инфраструктурных микросервисов;

5) провести исследование влияния работы авторизации на выполнения запросов между инфраструктурными сервисами.

Было проведено исследование времени выполнения запросов инфраструктурных сервисов как с включенной авторизацией, так и с выключенной. Исследование показало, что внедрение в систему инфраструктурных сервисов авторизации существенного влияния на работу не оказало.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Кравченко Д. А.* Микросервисная архитектура // Интерактивная наука. — 2022. — № 4. — С. 69.
2. *Mateus-Coelho N.*, *Cruz-Cunha M.*, *Ferreira L. G.* Security in Microservices Architectures // Procedia Computer Science. — 2021. — Т. 181. — С. 1225—1236. — ISSN 1877-0509. — DOI: https://doi.org/10.1016/j.procs.2021.01.320. — URL: https://www.sciencedirect.com/science/article/pii/S1877050921003719 ; (Дата обращения: 10.02.2025).
3. *Epping M.*, *Morowczynski M.* Authentication and Authorization // IDPro Body of Knowledge. — 2021. — Сент. — Т. 1. — DOI: 10.55621/idpro.78.
4. *Shaikh S.*, *Mane S.* Authentic techniques of authentication in microservices // International Journal of Current Advanced Research. — 2017. — Апр. — Т. 6. — С. 3342—3345. — DOI: 10.24327/ijcar.2017.3345.0267.
5. The OAuth 2.0 Authorization Framework. — URL: https://datatracker.ietf.org/doc/html/rfc6749#section-2.1 ; (Дата обращения: 10.02.2025).
6. *Khedrane A.*, *Salmi H.*, *Abdelhamid B.* Securing Web APIs with OAuth 2.0. — 2014. — Май.
7. *Li W.*, *Mitchell C.* User Access Privacy in OAuth 2.0 and OpenID Connect //. — 09.2020. — С. 664—6732. — DOI: 10.1109/EuroSPW51379.2020.00095.
8. *Ferdous M. S.* User-controlled Identity Management Systems using mobile devices : дис. ... канд. / Ferdous Md. Sadek. — 06.2015. — DOI: 10.13140/RG.2.1.3905.3287.

ПРИЛОЖЕНИЕ А

Презентация