



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:
«Система авторизации инфраструктурных сервисов»

Студент ИУ7-82Б
(Группа)

(Подпись, дата)

Васильев А. И.
(И. О. Фамилия)

Руководитель ВКР

(Подпись, дата)

Клорикьян П. В.
(И. О. Фамилия)

Нормоконтролер

(Подпись, дата)

(И. О. Фамилия)

2025 г.

РЕФЕРАТ

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

СОДЕРЖАНИЕ

РЕФЕРАТ	5
1 Технологический раздел	7
1.1 Развертывание k8s кластера	7
1.2 Реализация idP сервиса	13
1.3 Реализация клиента к idP	22
1.4 Тестирование программного обеспечения	30
2 Исследовательский раздел	33
2.1 Описание проводимого исследования	33
2.2 Технические характеристики устройства	35
2.3 Полученные результаты	36
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	37
ПРИЛОЖЕНИЕ А Презентация	38

1 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 1.1 приведена структура реализованного проекта.

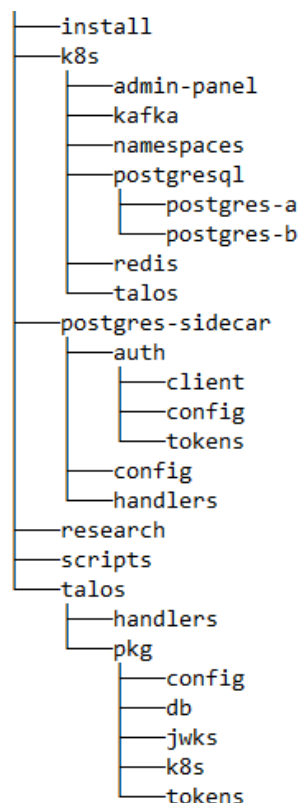


Рисунок 1.1 – Структура проекта

1.1 Развертывание k8s кластера

В листинге 1.1 приведен скрипт развертывания k8s сервисов.

Листинг 1.1 – 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/perpetualg0d/bmstu-diploma/talos:latest
./talos
docker push ghcr.io/perpetualg0d/bmstu-diploma/talos:latest
k3d image import ghcr.io/perpetualg0d/bmstu-diploma/talos:latest
-c bmstucluster --keep-tools

# run sidecar code in sidecar container:
docker build -t
  ghcr.io/perpetualg0d/bmstu-diploma/postgres-sidecar:latest
./postgres-sidecar
docker push
  ghcr.io/perpetualg0d/bmstu-diploma/postgres-sidecar:latest
k3d image import
  ghcr.io/perpetualg0d/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 1.2 и 1.3 приведен пример конфигурации сервиса вместе с сайдкармом в одном поде, а также конфигурация развертывания сервиса idP. Листинг 1.2 – Конфигурация развертывания PostgreSQL сервиса с сайдкармом

```

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/perpetualg0d/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 1.3 – 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/perpetualg0d/bmstu-diploma/talos:latest
          ports:
            - containerPort: 8080
      resources:
        limits:
          memory: "128Mi"
          cpu: "100m"

```

1.2 Реализация idP сервиса

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

В листинге 1.4 приведена SQL схема *infra2infra* и таблица для хранения прав.

Листинг 1.4 – 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 1.5–1.6 приведен способ генерации сертификата, основанного на rsa ключе, а также структура клеймов токена.

Листинг 1.5 – 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 (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}}
}

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 1.6 – 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 1.7–1.9 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 1.7 – 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 1.8 – 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 1.9 – 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 issuer: %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)
        }
    }, nil
}

```

```

        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 1.10–1.11 приведена реализация получения публичного k8s ключа сертификата проверки подписи k8s токена для последующего обмена на новый токен, который будет выпущен уже сервисом idP.

Листинг 1.10 – 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 1.11 – 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
}

```

1.3 Реализация клиента к idP

В листингах 1.12–1.13 приведена реализация получения публичного сертификата idP и фонового получения токена для проверки токена входящего запроса.

Листинг 1.12 – 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 1.13 – 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 1.14

Листинг 1.14 – 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(),
})
}
}

```

1.4 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 1.15.

Листинг 1.15 – Тест для проверки k8s токена

```

func Test_Verify(t *testing.T) {
    // Arrange
    token := "eyJhbGciOiJSUzI1NiIsImtpZCI6Ikp2UFNpY1pISzk1Y1hGRjhKSjJlY25MWHZRZ29aV25lMjB3UUV9IYUu5TmsifQ.eyJhdWQiOi0lsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiLCJrM3MiXSwiZmxhZCI6Imh0dHBzOi8va3ViZXJ1ZXRlcj5kZWZhdWx0LnN2YyIsImp0aSI6ImM0ZWUzNWZmLTZmZWYtNDNmYi1iNDYyLTJiNjk0OTM5MTdjMCI9Imt1YmVybWV0ZXMuZW8iOmsibmFtZCI6ImNwYWNlIjoicG9zdGdyZXMtYSIsIm5vZGU0nsibmFtZSI6ImszZC1ibXN0dWNsdXN0ZXItc2VydmVyLTAiLCJ1aWQiOi0iI3YzEwM2N1OC050TA2LTQ3NWt0GM5Ni1jNGZiZjIyOWM1YTAifSwicG9kIjp7Im5hbWUiOiJwb3N0Z3Jlcy1hLTU3OTRmY2I1ZjctcWI5em0iLCJ1aWQiOi0iI5MTE3NDl1Ny01NTFiLTQ2M2UtYmNmZC03ZjEyNGFhZTgxNWUifSwic2VydmVjZWZjY291bnQiOi0nsibmFtZSI6ImRlZmF1bHQiLCJ1aWQ

```

```

i0iIyNDhkNDE3Mi0xZDdlLTRiMWEtYmRlNS05NzAyN2FiNDFlMmQifSwi
d2FybmFmdGVyIjoxNzQ4MTE4NzYzfSwibmJmIjoxNzQ4MTE1MTU2LCJzd
WIi0iJzeXN0ZW06c2VydmljZWJjY291bnQ6cG9zdGdyZXMtYTpkZWZhdW
x0In0.spCWGajmAAENHgK5zG5NTX2dz82S0gunARu9-ncvurmV5XqEKOK
SEypC9Rm3ap2Wfw0ei0zm4-0Hbi3fdYZUZkSDZ0-HLSHxPZcQoGvVjiKD
OAIrDGwNb8bNVqkPPyy1Q8K5cD1anJwPiXYFzIQ4ZKDu_Ikp5ajkA3KYp
ZUmPLFQ3a09k8ycTpvdnjSzCfIdEaUi04jF1GJWEYKGo9XuY0VVjyyjwJ
Add0j6Ry0aIJzJzQTjS-IUs_dL7XVQUSlxp4mZvhLhnrhiL6uU59tX1QV
tliZ3Mgk03XN_F2G5kuoigFN0NQp8EpiCeC2-e9T-rchZ3MR8KbPp8lgs
4SA1iA"

```

```

jwk := JWK{
  N: 'xItwcttR4qVTD4bBfsUDgpICFnoBk1H8qyN3jSVemH1wlPyn6CLn2
aUmjQHW25f2LcraZr1_t7l0ogmar46Gn7uyYGBEtIsNnjvvoAUVbmd8vI
hPJi9flzDjJys4CEjefo1YFooD4YfqDei0GEYG2TYy42m03TR603--47P
bLIyZ2cbmHTwU-t_apqc3NUs0Sd6_gjDb0hrX0cFl0vBfL-J-3XEe4Zxe
w7-qDnjQGIKdSEgS-v-wCwr30iqK9yDfH09cHUtRNirLb4dybe0h3_vBM
MLCVpKtH6GonDEVyRv7qJCigEinpHB78Uq0PAb_l8S0Hougk2qp8-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 сайдкарром, приведена реализация idP сервиса, а также пример тестирования сервиса idP.

2 Исследовательский раздел

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

2.1 Описание проводимого исследования

На листингах 2.1–2.2 приведена реализация запроса к инфраструктурному сервису, а также запуск выполнения фиксированного количества запросов параллельно.

Листинг 2.1 – 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 2.2 – 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),
    })
}
}

```

2.2 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.

2.3 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 2.1.

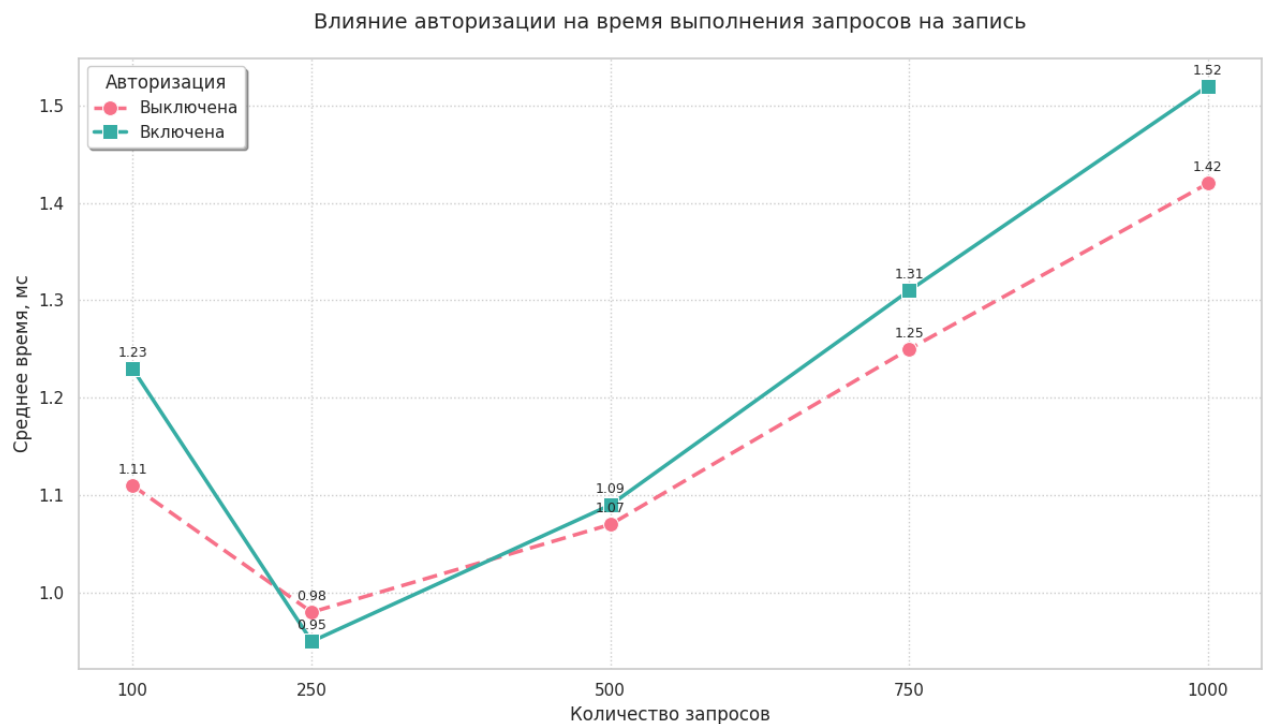


Рисунок 2.1 – Результаты исследования

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.

ПРИЛОЖЕНИЕ А

Презентация