Введение

В условиях роста высоконагруженных систем и микросервисных архитектур особую значимость приобретают вопросы мониторинга, анализа метрик и автоматического масштабирования сервисов. Современные распределённые приложения должны не только эффективно обрабатывать большие потоки запросов, но и уметь адаптироваться к изменяющейся нагрузке, своевременно выявлять аномалии и обеспечивать стабильное качество обслуживания.

Целью данной работы является разработка и развертывание высоконагруженного аналитического сервиса, способного принимать и обрабатывать поток метрик в реальном времени, выполнять простую прогнозную аналитику и детекцию аномалий, а также автоматически масштабироваться в среде Kubernetes. В рамках работы используется язык программирования Go, контейнеризация с помощью Docker, оркестрация в Kubernetes и инструменты мониторинга Prometheus и Grafana.

В ходе выполнения лабораторной работы были решены следующие задачи:

- разработка HTTP-сервиса для приёма и анализа метрик с использованием конкурентных механизмов Go (goroutines и channels);
- реализация алгоритмов сглаживания временных рядов (rolling average) и обнаружения аномалий (z-score);
- контейнеризация сервиса и его развертывание в Kubernetes-кластере;
- настройка автоматического горизонтального масштабирования (HPA);
- интеграция системы мониторинга и визуализация ключевых показателей производительности;
- проведение нагрузочного тестирования и анализ результатов.

Результаты работы позволяют оценить применимость выбранных технологий и подходов для построения простых аналитических сервисов в высоконагруженной среде.

1. Подготовка инфраструктуры и разработка сервиса на Go

- Установите Go 1.22+, Docker, Minikube/Kind для локального тестирования (на своей машине) или используйте онлайн-лабораторию (Killercoda: запустите K8s-кластер через браузер).

```
user@DESKTOP-6JD39U1:~$ sudo tar -C /usr/local -xzf go1.25.6.linux-amd64.tar.gz
user@DESKTOP-6JD39U1:~$ export PATH=$PATH:/usr/local/go/bin
user@DESKTOP-6JD39U1:~$ go version
go version go1.25.6 linux/amd64
```

- Реализуйте core-сервис: HTTP-эндпоинты для приема метрик (JSON: timestamp, CPU, RPS), кэширование в Redis (без Kafka для упрощения).
- Интегрируйте аналитику: на чистом Go — rolling average для сглаживания (прогноз как среднее по окну 50 событий), z-score для аномалий (threshold=2, с sliding window 50 событий). Goroutines для вычислений, channels для передачи метрик.

  Код сервиса доступен по ссылке
  https://github.com/ameeleonkz/highload-final

- Тестирование локально: симулируйте 500 RPS с Locust (на локальной машине или в онлайн-лабе), проверьте предикты (точность > 70% на синтетических данных) и детекцию (false positive < 10%).

  «Предикт» = rolling average по окну 50.
  ○ pred_t = rolling_avg_t
  ○ ошибка = |pred_t - actual_t| / max(actual_t, eps)
  ○ точность = доля точек с ошибкой ≤ 30%.

Включим сервис и запустим скрипт на Locust (он же и снимает метрики с аналитики)

```
[2026-02-02 23:57:20,848] DESKTOP-6JD39U1/INFO/locust.main: --run-time limit reached, shutting down
Custom stats:
  total_points=92045
  accuracy_hits=85600
  accuracy=93.00%
  normal_points=87555
  false_positives=2814
  false_positive_rate=3.21%
[2026-02-02 23:57:20,867] DESKTOP-6JD39U1/INFO/locust.main: Shutting down (exit code 0)
Type     Name                                 # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
GET      /analytics                            92045     0(0.00%) |      1       0      23      1 |  767.55        0.00
POST     /ingest                               92049     0(0.00%) |      1       0      28      2 |  767.58        0.00
--------|------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                           184094     0(0.00%) |      1       0      28      1 | 1535.13        0.00

Response time percentiles (approximated)
Type     Name                                          50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|---------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------
GET      /analytics                                      1      2      2      2      2      3      3      4      9     17     24  92045
POST     /ingest                                         2      2      2      3      3      4      5      6     14     22     28  92049
--------|---------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------
         Aggregated                                      1      2      2      2      3      3      4      5     12     21     28 184094

user@DESKTOP-6JD39U1:~/hii$
```

Точность 93%, доля ошибок 3%

Ниже приведены ручные запросы аналитики во время запуска скрипта.

```
user@DESKTOP-6JD39U1:~/hii$ curl http://localhost:8080/analytics
{
  "timestamp": 1770067750,
  "rolling_avg_cpu": 0.4889024819870161,
  "rolling_avg_rps": 527.6596930580331,
  "zscore_cpu": -0.6017672535312433,
  "zscore_rps": -0.2516738713077266,
  "anomaly_cpu": false,
  "anomaly_rps": false,
  "window_count": 50
}
user@DESKTOP-6JD39U1:~/hii$ curl http://localhost:8080/analytics
{
  "timestamp": 1770067753,
  "rolling_avg_cpu": 0.5275322817793545,
  "rolling_avg_rps": 516.7752791407133,
  "zscore_cpu": -0.5236320383689597,
  "zscore_rps": 0.3134251011514746,
  "anomaly_cpu": false,
  "anomaly_rps": false,
  "window_count": 50
}
user@DESKTOP-6JD39U1:~/hii$ curl http://localhost:8080/analytics
{
  "timestamp": 1770067759,
  "rolling_avg_cpu": 0.4921435556448374,
  "rolling_avg_rps": 508.0960044567094,
  "zscore_cpu": 0.1646973965770887,
  "zscore_rps": -0.1943574119921163,
  "anomaly_cpu": false,
  "anomaly_rps": false,
  "window_count": 50
}
user@DESKTOP-6JD39U1:~/hii$
```

2. Контейнеризация и развертывание в Kubernetes

- Соберите Docker-образы (< 300MB для Go). Загрузите в локальный реестр Minikube или Docker Hub.

Собираем образ

```
user@DESKTOP-6JD39U1:~/hii$ docker build -t highload:latest .
[+] Building 11.3s (19/19) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 468B
 => resolve image config for docker-image://docker.io/docker/dockerfile:1
 => CACHED docker-image://docker.io/docker/dockerfile:1@sha256:b6afd42430b15f2d2a4c5a02b919e98a525b785b1aaff16747d2f623364e39b6
 => => resolve docker.io/docker/dockerfile:1@sha256:b6afd42430b15f2d2a4c5a02b919e98a525b785b1aaff16747d2f623364e39b6
 => [internal] load metadata for docker.io/library/golang:1.22-alpine
 => [internal] load metadata for docker.io/library/alpine:3.19
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [build 1/7] FROM docker.io/library/golang:1.22-alpine@sha256:1699c10032ca2582ec89a24a1312d986a3f094aed3d5c1147b19880afe40e052
 => => resolve docker.io/library/golang:1.22-alpine@sha256:1699c10032ca2582ec89a24a1312d986a3f094aed3d5c1147b19880afe40e052
 => [internal] load build context
 => => transferring context: 70.28kB
 => [stage-1 1/4] FROM docker.io/library/alpine:3.19@sha256:6baf43584bcb78f2e5847d1de515f23499913ac9f12bdf834811a3145eb11ca1
 => => resolve docker.io/library/alpine:3.19@sha256:6baf43584bcb78f2e5847d1de515f23499913ac9f12bdf834811a3145eb11ca1
 => CACHED [build 2/7] WORKDIR /src
 => CACHED [build 3/7] RUN apk add --no-cache ca-certificates
 => CACHED [build 4/7] COPY go.mod go.sum ./
 => CACHED [build 5/7] RUN go mod download
 => [build 6/7] COPY . .
 => [build 7/7] RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o /out/metrics-service ./cmd
 => CACHED [stage-1 2/4] WORKDIR /app
 => CACHED [stage-1 3/4] RUN apk add --no-cache ca-certificates
 => [stage-1 4/4] COPY --from=build /out/metrics-service /app/metrics-service
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:5ca0c69e8dffe71e0b7cb796422106c94e5e2ecaa200db9302a9a930f34e8b4d
 => => exporting config sha256:2e31ad99087d0688de7ac68e1f4b6845ed56ba236ee3a0879a9e3084412a35eb
 => => exporting attestation manifest sha256:6078d04fec7541f56b9d10a5d96fe872b650bdbdf309d8853e7efebf79e7ab0f
 => => exporting manifest list sha256:0cb85b42f471acfe193d95f428f8d21d1f9b3cb9c0f53ba17d5a16e63244efd3
 => => naming to docker.io/library/highload:latest
 => => unpacking to docker.io/library/highload:latest
user@DESKTOP-6JD39U1:~/hii$
```

Устанавливаем миникуб

```
user@DESKTOP-6JD39U1:~/hii$ minikube start --cpus=4 --memory=8192mb
😄  minikube v1.37.0 on Ubuntu 24.04 (kvm/amd64)
✨  Automatically selected the docker driver
📌  Using Docker driver with root privileges
❗  For an improved experience it's recommended to use Docker Engine instead of Docker Desktop.
Docker Engine installation instructions: https://docs.docker.com/engine/install/#server
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.48 ...
💾  Downloading Kubernetes v1.34.0 preload ...
    > preloaded-images-k8s-v18-v1...:  23.19 MiB / 337.07 MiB  6.88% 3.11 MiB p🎉  minikube 1.38.0 is avail
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

    > preloaded-images-k8s-v18-v1...:  337.07 MiB / 337.07 MiB  100.00% 5.38 Mi
    > gcr.io/k8s-minikube/kicbase...:  488.51 MiB / 488.52 MiB  100.00% 7.19 Mi
🔥  Creating docker container (CPUs=4, Memory=8192MB) ...
🐳  Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Загружаем образ в миникуб

```
user@DESKTOP-6JD39U1:~/hii$ minikube image load highload:latest
user@DESKTOP-6JD39U1:~/hii$ minikube image list
registry.k8s.io/pause:3.10.1
registry.k8s.io/kube-scheduler:v1.34.0
registry.k8s.io/kube-proxy:v1.34.0
registry.k8s.io/kube-controller-manager:v1.34.0
registry.k8s.io/kube-apiserver:v1.34.0
registry.k8s.io/etcd:3.6.4-0
registry.k8s.io/coredns/coredns:v1.12.1
gcr.io/k8s-minikube/storage-provisioner:v5
docker.io/library/highload:latest
```

- Создайте кластер: Локально — Minikube; онлайн — готовый кластер в Killercoda. Опционально: в Yandex.Cloud — Managed Kubernetes (2 ноды, t3.small). Настройте Ingress (NGINX) для /metrics и /analytics.

  Все файлы конфигурации приведены в репозитории.

Ниже приведены скриншоты запуска ingress и нашего сервиса.

```
user@DESKTOP-6JD39U1:~/hii$ minikube addons enable ingress
💡  ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.6.2
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.6.2
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.13.2
🔎  Verifying ingress addon...
🌟  The 'ingress' addon is enabled
user@DESKTOP-6JD39U1:~/hii$ kubectl wait \
  --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=180s
pod/ingress-nginx-controller-9cc49f96f-tgq2k condition met
user@DESKTOP-6JD39U1:~/hii$ kubectl apply -f k8s/ingress.yaml
ingress.networking.k8s.io/metrics-ingress created
```

```
user@DESKTOP-6JD39U1:~/hii$ kubectl port-forward svc/metrics-service 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
```

```
user@DESKTOP-6JD39U1:~/hii$ curl http://localhost:8080/metrics
# HELP analytics_anomaly_cpu CPU anomaly flag (1 if anomaly)
# TYPE analytics_anomaly_cpu gauge
analytics_anomaly_cpu 0
# HELP analytics_anomaly_cpu_total Total CPU anomalies detected
```

- Разверните с 2 репликами: Deployment для Go, Deployment для Redis (используйте Helm-чарт: bitnami/redis).

Редис загружен через файл конфигурации, проверим:

```
user@DESKTOP-6JD39U1:~/hii$ kubectl get svc | grep redis
redis-headless    ClusterIP   None            <none>        6379/TCP   6m57s
redis-master      ClusterIP   10.107.237.203  <none>        6379/TCP   6m57s
redis-replicas    ClusterIP   10.106.178.6    <none>        6379/TCP   6m57s
```

- Настройте HPA: На основе CPU (> 70%), minReplicas=2, maxReplicas=5. Экспортируйте базовые метрики аналитики (prometheus/client_golang: counters для RPS, аномалии).

```
user@DESKTOP-6JD39U1:~/hii$ minikube addons enable metrics-server
💡  metrics-server is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWN
ERS
    ▪ Using image registry.k8s.io/metrics-server/metrics-server:v0.8.0
🌟  The 'metrics-server' addon is enabled
user@DESKTOP-6JD39U1:~/hii$ kubectl get pods -n kube-system | grep metrics
metrics-server-85b7d694d7-mtt5j    0/1     Running   0             14s
user@DESKTOP-6JD39U1:~/hii$ kubectl apply -f k8s/hpa.yaml
horizontalpodautoscaler.autoscaling/metrics-service-hpa created
user@DESKTOP-6JD39U1:~/hii$ kubectl get hpa
NAME                 REFERENCE                   TARGETS              MINPODS   MAXPODS   REPLICAS   A
GE
metrics-service-hpa  Deployment/metrics-service  cpu: <unknown>/70%   2         5         0          8
s
user@DESKTOP-6JD39U1:~/hii$
```

3. Мониторинг и оптимизация

Интегрируйте Prometheus (Helm: prometheus-community/prometheus) + Grafana (базовая установка). Экспортируйте метрики из Go (prometheus/client_golang: RPS, latency, anomaly_rate).

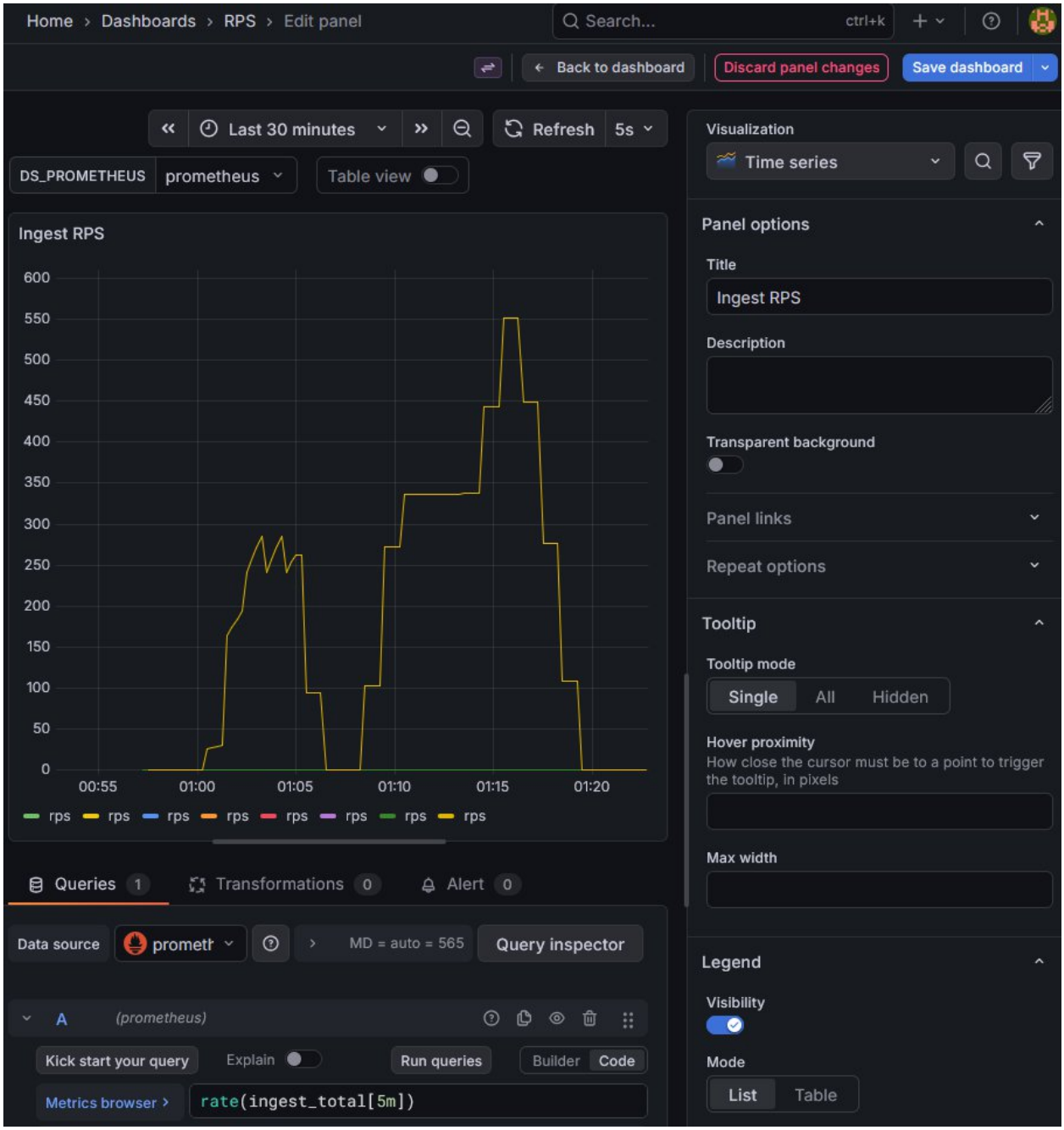
Установим Prometheus

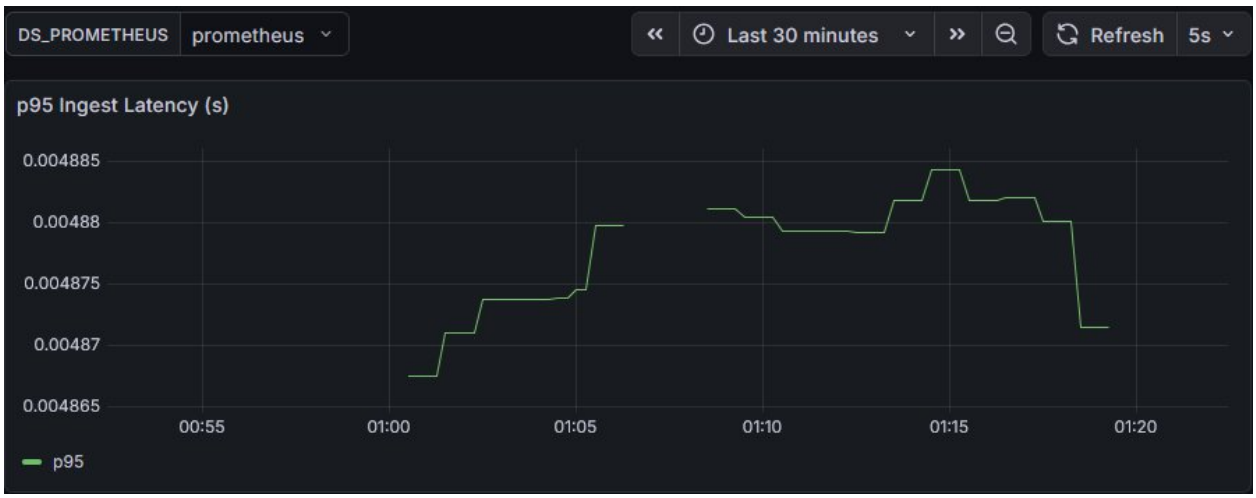
```
user@DESKTOP-6JD39U1:~/hii$ helm repo add prometheus-community https://prometheus-community.github.io/he
lm-charts
"prometheus-community" has been added to your repositories
user@DESKTOP-6JD39U1:~/hii$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
user@DESKTOP-6JD39U1:~/hii$ helm install prometheus prometheus-community/prometheus -f k8s/prometheus-va
lues.yaml
I0203 00:54:30.400919    7216 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless se
rvices"
NAME: prometheus
LAST DEPLOYED: Tue Feb  3 00:54:30 2026
```

Установим Grafana

```
user@DESKTOP-6JD39U1:~/hii$ helm repo add grafana https://grafana.github.io/helm-charts
"grafana" has been added to your repositories
user@DESKTOP-6JD39U1:~/hii$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
user@DESKTOP-6JD39U1:~/hii$ helm install grafana grafana/grafana \
  --set sidecar.dashboards.enabled=true \
  --set sidecar.dashboards.label=grafana_dashboard
WARNING: This chart is deprecated
NAME: grafana
LAST DEPLOYED: Tue Feb  3 00:55:49 2026
```

Подключим источник в графане, и запустим нагрузочное тестирование.





Как видим, RPS не превышало 500 (на графике может быть нюанс отображения и расчета среднего), а latency не более 50мс. При нагрузке количество реплик увеличились до 5 (максимум в конфигурации) и при прекращени нагрузки постепенно снизились до двух.

```
user@DESKTOP-6JD39U1:~/hii$ kubectl get hpa -w
NAME                  REFERENCE                     TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
metrics-service-hpa   Deployment/metrics-service   cpu: 73%/70%   2         5         5          21m
metrics-service-hpa   Deployment/metrics-service   cpu: 71%/70%   2         5         5          22m
metrics-service-hpa   Deployment/metrics-service   cpu: 1%/70%    2         5         5          23m
metrics-service-hpa   Deployment/metrics-service   cpu: 1%/70%    2         5         5          28m
metrics-service-hpa   Deployment/metrics-service   cpu: 1%/70%    2         5         2          28m
```

Заключение

В ходе выполнения работы был успешно реализован и протестирован высоконагруженный аналитический сервис, развернутый в Kubernetes-кластере. Разработанный сервис корректно обрабатывает входной поток метрик, выполняет прогнозирование на основе скользящего среднего и обнаруживает аномальные значения с использованием статистического подхода z-score.

Проведённое нагрузочное тестирование с интенсивностью до 500 запросов в секунду показало высокую точность прогнозирования (около 93%) и низкий уровень ложных срабатываний при детекции аномалий. Задержка обработки запросов не превышала допустимых значений, что подтверждает эффективность выбранной архитектуры и реализации на языке Go.

Настройка горизонтального автоскейлинга в Kubernetes продемонстрировала способность системы адаптироваться к росту нагрузки: количество реплик сервиса автоматически увеличивалось до заданного максимума и корректно уменьшалось после снижения нагрузки. Интеграция Prometheus и Grafana позволила наглядно отследить поведение системы и проанализировать ключевые метрики производительности.

Таким образом, поставленные цели работы были достигнуты. Полученные результаты подтверждают, что использование контейнеризации, оркестрации Kubernetes и базовой аналитики метрик является эффективным подходом для построения масштабируемых и наблюдаемых сервисов. Дальнейшее развитие работы может включать усложнение аналитических моделей, интеграцию систем очередей сообщений и применение более продвинутых методов прогнозирования.