Содержание

[1 Техническое задание 4](#_Toc507418945)

[2 Теоретическая часть 4](#_Toc507418946)

[2.1 Выявление заинтересованных сторон и их интересов 4](#_Toc507418947)

[2.2 Docker 5](#_Toc507418948)

[2.3 Kubernetes 6](#_Toc507418949)

[2.4 Prometheus 7](#_Toc507418950)

[2.5 Выбор технических решений, удовлетворяющих интересам ЗС 9](#_Toc507418951)

[2.6 Структура проекта 9](#_Toc507418952)

[2.6.1 Вычислительный кластер на базе Spark. 9](#_Toc507418953)

[2.6.2 Prometheus Operator 10](#_Toc507418954)

[2.6.3 Prometheus Node-Exporter 10](#_Toc507418955)

[2.6.4 Kube-state-metrics 10](#_Toc507418956)

[2.6.5 Grafana 10](#_Toc507418957)

[2.7 Диаграмма компонентов 10](#_Toc507418958)

[3 Технологическая часть 11](#_Toc507418959)

[3.1 Запуск вычислительного кластера с мониторингом. 11](#_Toc507418960)

[3.2 Тестирование на корректность работы 12](#_Toc507418961)

[4 Выводы 13](#_Toc507418962)

[5 Список источников 14](#_Toc507418963)

# Техническое задание

Разработка мониторинга производительности вычислительного кластера на базе Spark с использованием технологии контейнеризации:

* изучить технологию контейнеризации Docker;
* изучить технологию для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями Kubernetes;
* изучить технологию мониторинга Prometheus;
* реализовать систему мониторинга вычислительного кластера на базе Spark с использованием Kubernetes и Prometheus;
* описать требования, конструкцию, особенности сборки и запуска в документации.

# Теоретическая часть

## Выявление заинтересованных сторон и их интересов

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица . Заинтересованные стороны и их интересы по отношению к системе

|  |  |
| --- | --- |
| **Заинтересованные стороны** | **Интересы заинтересованных сторон** |
| Системный администратор вычислительного кластера | С1 Быстрое и точное реагирование на аварийные ситуации  С2 Быстрая развертка вычислительного кластера с интегрированным мониторингом  С3 Возможность добавления в систему новых контейнеров без настройки мониторинга  С4 Триггерные предупреждения и уведомления  С5 Наглядность кластеров и планирование пропускной способности  С6 Удобные и содержательные таблицы показателей  С7 Проактивный мониторинг. |

## Docker

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Существуют сборки только для платформ x86-64 и ARM. Начиная с версии 1.6 возможно использование в ОС Windows. [1]

В состав программных средств входит демон — сервер контейнеров (запускается командой docker -d), клиентские средства, позволяющие из интерфейса командной строки управлять образами и контейнерами, а также API, позволяющий в стиле REST управлять контейнерами программно.

Демон обеспечивает полную изоляцию запускаемых на узле контейнеров на уровне файловой системы (у каждого контейнера собственная корневая файловая система), на уровне процессов (процессы имеют доступ только к собственной файловой системе контейнера, а ресурсы разделены средствами libcontainer), на уровне сети (каждый контейнер имеет доступ только к привязанному к нему сетевому пространству имён и соответствующим виртуальным сетевым интерфейсам).

Набор клиентских средств позволяет запускать процессы в новых контейнера, останавливать и запускать контейнеры, приостанавливать и возобновлять процессы в контейнерах. Серия команд позволяет осуществлять мониторинг запущенных процессов. Новые образы возможно создавать из специального сценарного файла (файл сценария носит название dockerfile), возможно записать все изменения, сделанные в контейнере в новый образ. Все команды могут работать как с docker-демоном локальной системы, так и с любым сервером Docker, доступным по сети. Кроме того, в интерфейсе командной строки встроены возможности по взаимодействию с публичным репозиторием Docker Hub, в котором размещены предварительно собранные образы контейнеров, что позволяет осуществить поиск образов среди размещённых в нём. Образы можно скачивать в локальную систему, возможно также отправить локально собранные образы в Docker Hub.

Docker имеет ряд основных преимуществ в сравнении с другими технологиями контейнеризации:

* Легкость в обновлении и поддержке образов
* Распределение ресурсов и анализ (более безопасный способ изоляции данных)
* Простая интеграция в проекты с Dockerfiles (самостоятельно настроенные пользовательские среды для каждого компонента системы).

## Kubernetes

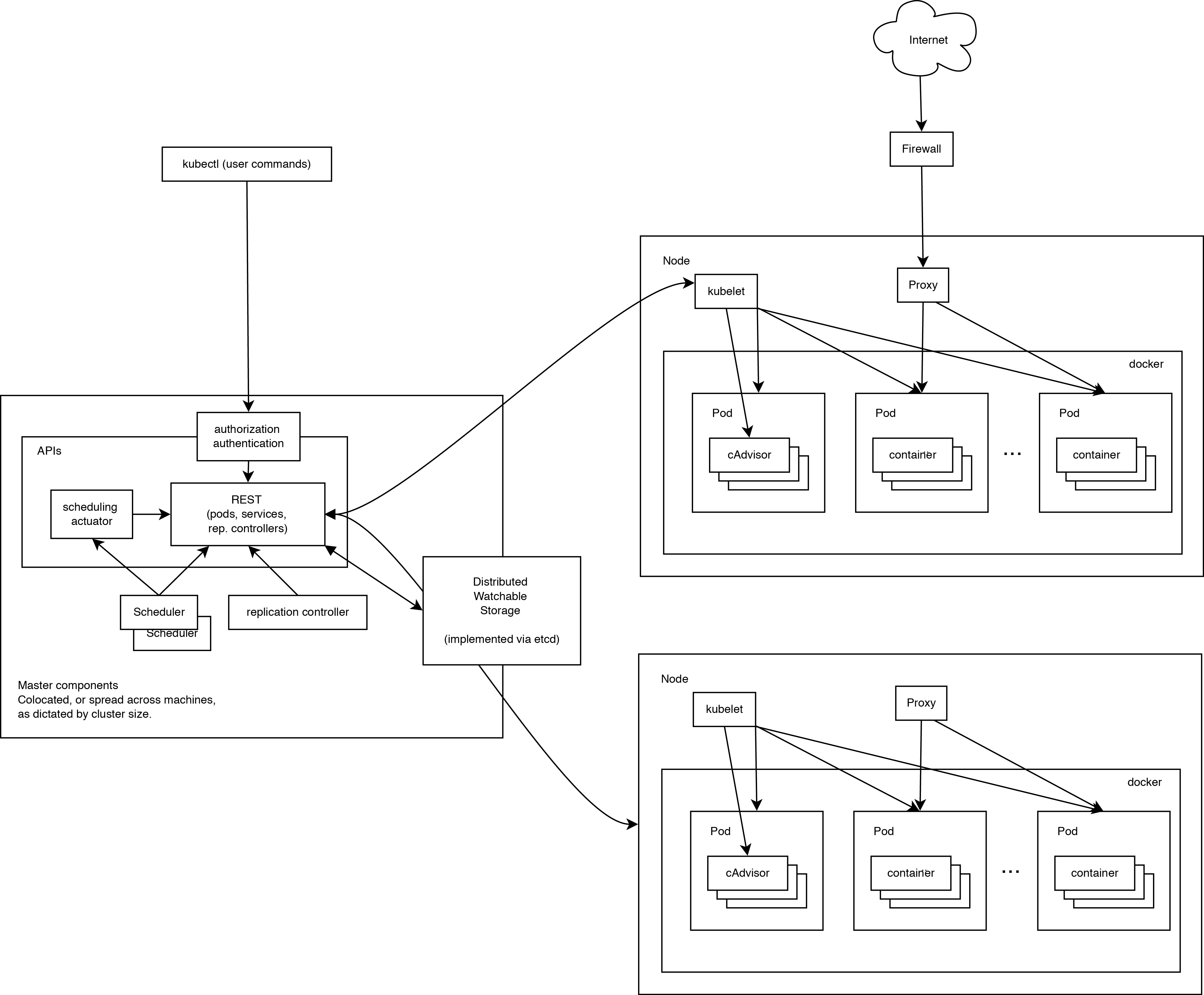
Kubernetes — открытое программное обеспечение для автоматизации развёртывания, масштабирования и управления контейнеризированными приложениями.

Основные концепции Kubernetes:

* Nodes (node.md): Физическая машина в кластере Kubernetes;
* Pods (pods.md): Группа контейнеров с общими разделами, запускаемых как единое целое;
* Replication Controllers (replication-controller.md): Контроллер гарантирует, что определенное количество «реплик» будут запущены в любой момент времени;
* Services (services.md): Абстракция, объединяющая логический объединённый набор pod и политику доступа к ним;
* Volumes (volumes.md): Директория с данными в ней, которая доступна в контейнере;
* Labels (labels.md): Пары ключ/значение, которые прикрепляются к объектам. Они могут быть использованы для создания и выбора наборов объектов;
* Kubectl Command Line Interface (kubectl.md): Интерфейс командной строки для управления Kubernetes;

Работающий кластер Kubernetes включает в себя агента, запущенного на нодах (kubelet) и компоненты мастера (APIs, scheduler, etc), поверх решения с распределённым хранилищем (рисунок 1).

На каждой ноде Kubernetes запускаются сервисы, необходимые для управления нодой со стороны мастера и для запуска приложений, а также запускается Docker, который обеспечивает загрузку образов и запуск контейнеров [2].



**Рисунок 1. Архитектура Kubernetes**

Основными преимуществами Kubernetes является хорошая масштабируемость и автоматизация.

## Prometheus

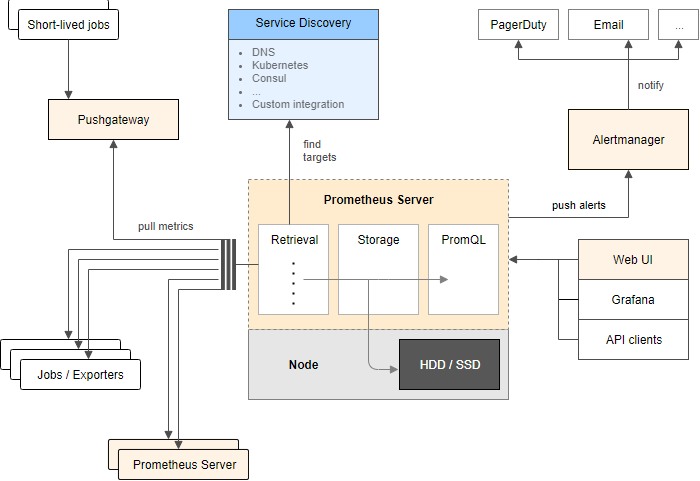
Prometheus — это инструментарий для мониторинга и оповещения систем с открытым исходным кодом. Основные особенности Prometheus:

* многомерная модель данных с данными временных рядов, идентифицированными метрическим именем и парами ключ/значение;
* гибкий язык запросов;
* одиночные серверные узлы являются автономными;
* коллекция временных рядов происходит с помощью модели pull по HTTP;
* толкание временных рядов поддерживается через посреднический шлюз;
* цели обнаруживаются через обнаружение службы или статическую конфигурацию;
* несколько режимов графического отображения и поддержки панели инструментов.

Главный компонент всей системы — сервер Prometheus. Он работает автономно и сохраняет все данные в локальной базе данных. Обнаружение сервисов происходит автоматически. Это упрощает процедуру развёртывания: для наблюдения за одним сервисом не нужно разворачивать распределённую систему мониторинга; достаточно установить только сервер и необходимые компоненты для сбора и экспорта метрик.

Сбор метрик в Prometheus осуществляется с помощью механизма pull. Имеется также возможность сбора метрик с помощью механизма push (для этого используется специальный компонент pushgateway, который устанавливается отдельно) [3].

Подробная архитектура Prometheus представлена на рисунке 2.



**Рисунок 2. Архитектура Prometheus**

Основными преимуществами в сравнении с аналогами являются:

* простота развертывания;
* широкие возможности интеграции со сторонними приложениями и сервисами;
* удобный графический интерфейс для работы с метриками.

## Выбор технических решений, удовлетворяющих интересам ЗС

В таблице ниже представлены результаты выбора технических решений, позволяющие удовлетворить интересы заинтересованных сторон по отношению к системе.

Таблица . Технические решения, удовлетворяющие интересам ЗС

|  |  |
| --- | --- |
| **Интересы заинтересованных сторон** | **Технические решения** |
| С1 Быстрое и точное реагирование на аварийные ситуации  С2 Быстрая развертка вычислительного кластера с интегрированным мониторингом  С3 Возможность добавления в систему новых контейнеров без настройки мониторинга  С4 Триггерные предупреждения и уведомления  С5 Наглядность кластеров и планирование пропускной способности  С6 Удобные и содержательные таблицы показателей  С7 Проактивный мониторинг. | Будет использована технология Kubernetes для автоматизации развёртывания, масштабирования и управления контейнеризированными приложениями вычислительного кластера (Hadoop, Spark и другие).  Будет использован Kube-Prometheus для реализации мониторинга кластера Kubernetes.  Будет использован AlertManager как менеджер уведомления для своевременного обнаружения аварийных ситуаций, их предсказывания и анализа.  Будет использована графическая оболочка Grafana для визуализации входящих метрик. |

## Структура проекта

### Вычислительный кластер на базе Spark.

За основу вычислительного кластера возьмем уже реализованный кластер на Kubernetes. Исходный код доступен в репозитории GitHub.

Компоненты, входящие в вычислительный кластер:

* Spark v.2.2.1;
* Hive v.2.3.2;
* Zeppelin v.0.7.3;
* Hadoop 3.0.0;
* Traefik 1.5.2.

### Prometheus Operator

Данный оператор создает, настраивает и управляет экземплярами мониторинга Prometheus [4].

Оператор основывается на двух основных концепциях Kubernetes: «Ресурсы и контроллеры». Например, встроенный ресурс ReplicaSet позволяет пользователям задавать необходимое количество номеров для Pods, а контроллеры внутри Kubernetes гарантируют, что желаемый набор состояний в ресурсе ReplicaSet остается верным, создавая или удаляя запущенные Pods. В Kubernetes есть много фундаментальных контроллеров и ресурсов, которые работают таким образом, включая службы , развертывания и DaemonSet.

Оператор, включают в себя безопасную координацию обновлений приложений, настройку резервных копий на внешнее хранилище, обнаружение служб через собственные API-интерфейсы Kubernetes, конфигурацию сертификата TLS приложения и аварийное восстановление.

При разворачивании новой версии приложения, Kubernetes создаёт новый под и, когда этот под готов, уничтожает старый. Prometheus постоянно отслеживает Kubernetes API и, когда обнаруживает изменения, создаёт новую конфигурацию Prometheus, руководствуясь изменениями в службах (подах).

### Prometheus Node-Exporter

Prometheus экспортер для аппаратных и операционных показателей, выставленных unux ядрами, написанными на Go с подключаемыми метрическими коллекторами. [5]

### Kube-state-metrics

Служба, которая слушает сервер API Kubernetes и генерирует метрики о состоянии объектов. Она фокусируется на работоспособности различных объектов внутри, таких как развертывания, узлы и контейнеры.

### Grafana

Пакет для визуализации данных мониторинга, хранящихся в Prometheus [6].

## Диаграмма компонентов

На рисунке 3 представлена архитектура вычислительного кластера.

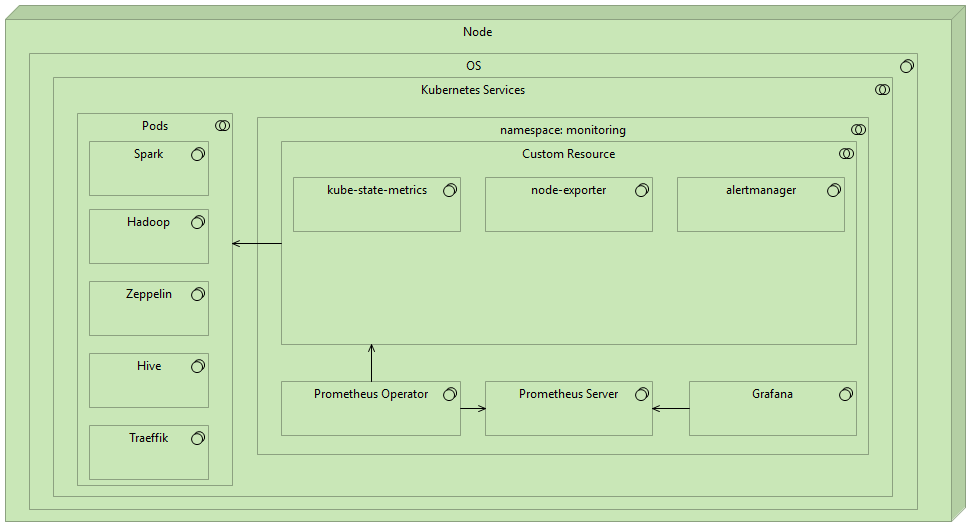


Рисунок 3. Архитектура мониторинга вычислительного кластера

Prometheus Operator использует Custom Resource Definition (CRD). Он разыскивает поды, соответствующие селектору лейблов (выбирает с помощью matchLabels), и создаёт целевой объект, с которого Prometheus будет собирать метрики.

Kube-state-Metrics собирает информацию о всех сервисах Kubernetes.

Node-Exporter собирает информацию о всех узлах, находящихся в кластере.

AlertManager получает от Prometheus Server сигналы, и в зависимости от них генерирует почтовые сообщения с предупреждениями.

# Технологическая часть

## Запуск вычислительного кластера с мониторингом.

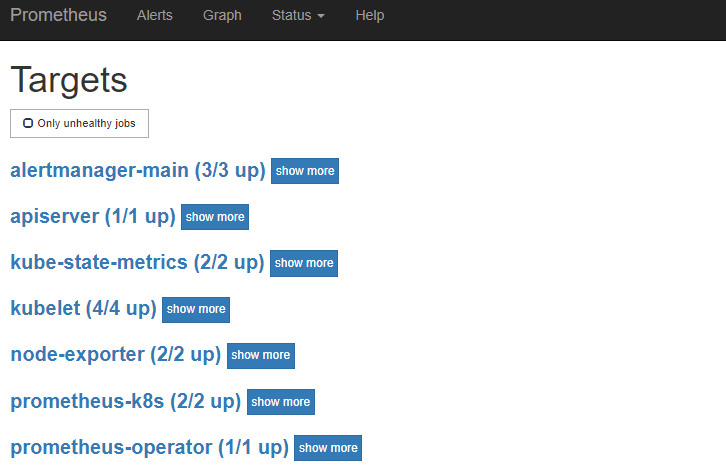
Исходный код проекта доступен в репозитории GitHub [7]. Подробная информация по сборке представлена в файле Readme.md.

## Тестирование на корректность работы

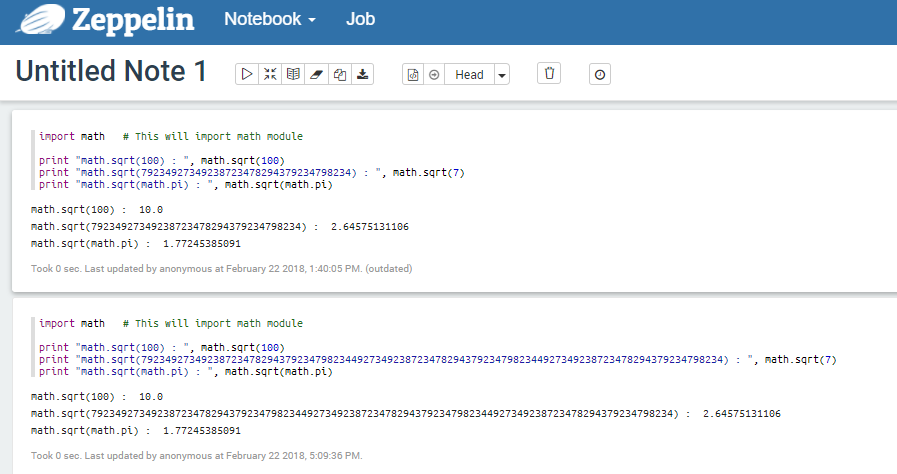
Тестирование системы было проведено на Google Kubernetes Engine. Информация по установке на GKE [8] приведена в файле Readme.md. На рисунках 4, 5, 6, 7 представлены результаты работы Grafana, Prometheus, Zeppelin и Google Kubernetes Engine.



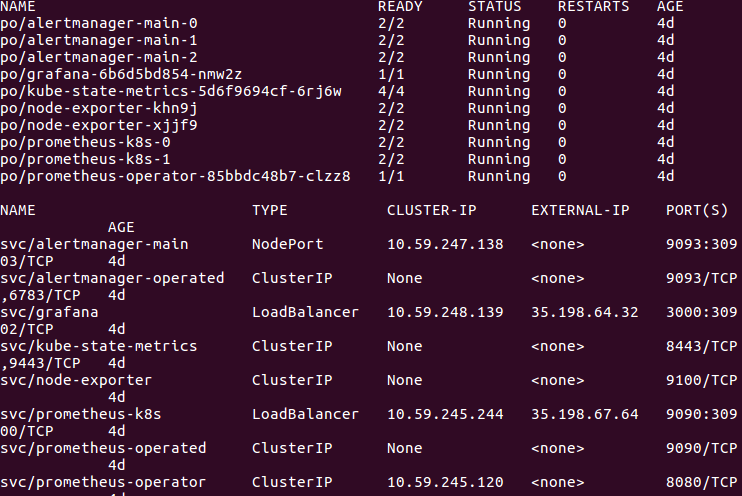
**Рисунок 4. Мониторинг всех узлов Kubernetes. Grafana.**



**Рисунок 5. Цели в Prometheus Server.**

****

**Рисунок 6. Тестирование Zeppelin**

****

**Рисунок 7. Деплой на Google Kubernetes Engine**

# Выводы

В ходе курсовой работы были изучены технологии контейнеризации (Docker), автоматизации развёртывания, масштабирования и управления контейнеризированными приложениями (Kubernetes), а также инструментарий для мониторинга и оповещения систем (Prometheus, Grafana), выявлены их преимущества.

Кроме того, в ходе курсовой работы с использованием технологии Prometheus Operator был спроектирован, реализован и протестирован Kube Prometheus, позволяющий проводить мониторинг любого кластера на Kubernetes, включая вычислительные. Было проведено тестирование на GKE.

# Список источников

[1] Документация по технологии Docker: [Электронный ресурс]. Режим доступа: https://docs.docker.com/.

[2] Документация по технологии Kubernetes: [Электронный ресурс]. Режим доступа: https://kubernetes.io/docs/home/supported-doc-versions/.

[3] Документация по технологии Prometheus: [Электронный ресурс]. Режим доступа: https://prometheus.io/docs/introduction/overview/.

[4] GitHub - Prometheus Operator [Электронный ресурс]. Режим доступа: https://github.com/coreos/prometheus-operator

[5] GitHub - Node Exporter [Электронный ресурс]. Режим доступа: https://github.com/prometheus/node\_exporter

[6] Документация по технологии Grafana: [Электронный ресурс]. Режим доступа: http://docs.grafana.org/.

[7] GitHub - артефакты проекта: [Электронный ресурс]. Режим доступа: https://github.com/SannikovDmitry/k8s-prometheus-grafana-gke/

[8] Документация по Google Kubernetes Engine: [Электронный ресурс]. Режим доступа: https://cloud.google.com/compute/docs/

# 