Раздел 3 ИНТЕЛЛЕКТУАЛЬНЫЕ ТРАНСПОРТНЫЕ СИСТЕМЫ

УДК 004.42
DOI: 10.34046/aumsuomt105/39

DEVOPS И СОВРЕМЕННАЯ СЕРВЕРНАЯ АРХИТЕКТУРА

И.В. Родыгина, кандидат технических наук, доцент
А.А. Биденко, студент
Musab Bassam Zghoul, PhD, Assistant Professor

Неотъемлемой частью любого многопользовательского сервиса является его серверная часть. Хранение данных, логика авторизации, алгоритм требующие безопасности, чтобы ни один пользователь не мог найти уязвимость в системе. Все это возложено на бэкенд – серверную логику приложения. От безопасности, стабильности, непрерывного обновления серверного кода зависят без преувеличения тысячи и миллионы пользователей по всему миру. Обеспечить максимальные показатели вышеперечисленных свойств обязанность каждого разработчика.

Разработка любого приложения зависит от его архитектуры и архитектуры каждого компонента. Ранее настройка серверной среды и разработка бэкенда части сервиса были неразрывны. Но приложения становятся все сложнее, требования к системе все труднее в реализации. Настройка серверной среды, автоматизация процессов работы и обновления выделены в отдельную сферу деятельности и IT отрасль – DevOps. В мире существуют инструменты контейнеризации, автоматизации процессов, web-сервера и многое другое. И ни одна из этих технологий не используется по отдельности. Для разработчика программного обеспечения без соответствующих навыков в области DevOps является сложностью реализация грамотной настройки сервера.

Ключевые слова: сервер, архитектура, DevOps, разработка ПО, Docker, Caddy, SSL, TLS, Golang, GitHub Actions.

DEVOPS AND MODERN SERVER ARCHITECTURE

I.V. Rodygina, A. A. Bidenko, Musab Bassam Zghoul

An integral part of any multi–user service is its server part. Data storage, authorization logic, algorithm requiring security so that no user can find a vulnerability in the system. All this is assigned to the backend – the server logic of the application. Without exaggeration, thousands and millions of users around the world depend on security, stability, and continuous updating of the server code. It is the responsibility of each developer to ensure the maximum performance of the above properties.

The development of any application depends on its architecture and the architecture of each component. Previously, setting up the server environment and developing the backend of the service part were inseparable. But applications are becoming more complex, system requirements are becoming more difficult to implement. Setting up a server environment, automating work processes and updates are highlighted in a separate area of activity and the IT industry – DevOps. There are containerization tools, process automation tools, web servers and much more in the world. And none of these technologies are used in isolation. For a software developer without appropriate DevOps skills, it is difficult to implement a competent server configuration.

Keywords: server, architecture, DevOps, Software Development, Docker, Caddy, SSL, TLS, Golang, GitHub Actions.

Введение. Неотъемлемой частью любого многопользовательского сервиса является его серверная часть. Хранение данных, логика авторизации, алгоритм требующие безопасности, чтобы ни один пользователь не мог найти уязвимость в системе. Все это возложено на бэкенд–серверную логику приложения.

От безопасности, стабильности, непрерывного обновления серверного кода зависят миллионы пользователей по всему миру. Обеспечить максимальные показатели вышеперечисленных свойств обязанность каждого разработчика. Разработка любого приложения зависит от его архитектуры и архитектуры каждого компонента. Ранее настройка серверной среды и разработка бэкенда части сервиса были неразрывны. Но приложения становятся все сложнее, требования к системе все труднее в реализации. Настройка серверной среды, автоматизация процессов работы и обновления были выделены в отдельную сферу деятельности и IT отрасль – DevOps. Она отвечает за создание условий, при которых взлом и дестабилизация работы приложения будет все затруднительнее, а процессы работы и обновления все легче и комфортнее.

Но вместе с новыми требованиями появляются и новые технологии. В мире существуют инструменты контейнеризации, автоматизации процессов, web-сервера и многое другое. И ни одна из этих технологий не используется по отдельности. Для разработчика программного обеспечения без соответствующих навыков в области DevOps является сложностью реализация грамотной настройки сервера. Подобной работой занимаются сейчас отдельные специалисты или системные администраторы. Но число специалистов DevOps в России крайне мало, а их средняя оплата труда вызывает большие трудности при найме в малые и средние компании. Системные же администраторы больше решают проблемы на уровне аппаратного ПО и используют устаревшие технологии при настройке сервера.

Обзор существующих инструментов интеграции.

Docker – инструмент контейнеризации приложений. Контейнеры – упрощенные аналоги виртуальных машин. Они создают стабильную среду работы любого сервиса, а также упрощают процессы непрерывного обновления, мониторинга и перезапуска компонента приложения. Разработчик имеет возможность полного контроля работы приложения из кода. При смене среды запуска контейнера показатели его работы остаются неизменными [8,9].

Существуют различные аналоги инструментов непрерывной интеграции и непрерывного деплоинга, так называемые инструменты CI/CD [12]. Самыми популярными инструментами являются Gitlab CI и GitHub Actions [7]. Цель любых таких инструментов – дать возможность разработчику настраивать среду исполнения любых команд, позволяющих выполнить любые действия в автоматическом режиме, когда это необходимо. Простыми примерами являются автотестирование и автоматический деплоинг на сервер.

Web-сервер осуществляет прием http запросов и запросов иных протоколов клиент серверного общения и дальнейшего проксирования запросов на различные компоненты серверной среды. Редиректы, проксирование запросов, доступ к файловой структуре сервера – за все эти процедуры отвечает web-сервер. Любой доступ к серверной среде в обход настроек web-сервера для любого пользователя системы является небезопасным.

Проблемы серверной архитектуры. Генерация SSL(TLS) сертификатов безопасности, защищенное соединение является обязательным условием любого приложения. Браузеры понижают в поисковом ранжировании сайты с небезопасным соединением, а безопасность пользователей становится под угрозой. Некоторые хостинги предлагают услуги по автоматической генерации сертификатов, а также есть программа CertBot для их создания. Первое решение подходит только для статического хостинга, а второе сложно в реализации.

Работа нескольких контейнеров на одном сервере. Все программы указанные выше помогают легко решать проблемы при наличии одного компонента приложения, но при увеличении их числа становится все сложнее масштабировать и развивать всю систему. А сейчас все больше появляется необходимость у различных компаний в развитии тестовой среды исполнения своего кода – упомянутые выше технологии вообще изначально не создавались под подобную задачу.

Технологии для решения проблемы. Caddy – web-сервер с новым функционалом. Наиболее популярными web-серверами на сегодняшний день являются Nginx или Apache. Первый имеет гибкую настройку и высокую скорость исполнения. Он идеально подходит для глубокой настройки конфигурации для конкретного компонента приложения. Второй же завоевал популярность на рынке статичных хостингов за счет настройки на уровне директорий. Однако Caddy добавляет возможность простой настройки сразу для множества доменных имен, а также автоматически генерирует сертификаты безопасности, что решает часть из упомянутых выше проблем.

Golang – это не решение какой–то конкретной проблемы, но инструмент для создания своих собственных программных решений. Это язык программирования низкого уровня способной выполнять самые различные задачи. Если не существует готового решения или существующие решения не выполняют необходимые задачи, то этот язык программирования является одним из самых простых и универсальных для создания собственных решений.

Все вышеперечисленные технологии можно объединить в полноценный сервис, решающий проблемы серверной архитектуры. Безопасность, стабильность, непрерывность работы становятся основными целями разрабатываемого решения.

Технология контейнеризации отлично служит для создания среды исполнения отдельных приложений. Возможность управлять средой исполнения из Dockerfile отдельных компонентов

удобна для разработчиков любых сервисов. Размещение настроек работы элементов приложений в репозитории проектов является хорошим решением для управления среды исполнения.

Каждый компонент приложения, будь это бэкенд, фронтенд или что-то иное может нуждаться в уникальных настройках редиректов, проксирования и иных параметров. Размещение web-серверов внутри репозиториев отдельных компонентов сервисов также является удобным решением. Но инфраструктура контейнеров такова, что они хорошо работают только с одним параллельно запущенным процессом внутри. Если мы работаем с группой контейнеров в одном репозитории, то нам нужен инструмент контроля группы контейнеров и эту задачу выполняет консольная утилита docker–compose. Один репозиторий может содержать несколько Dockerfile для разных компонентов одной части приложения и один docker-compose.yml файл для их групповой настройки. Если же для работы части сервиса необходимы компоненты такие как, например, база данных, сервисы очередей и т. д., то их настройку без отдельных Dockerfile можно выполнять в docker-compose.yml файле.

Для автоматического тестирования и деплоинга на сервер существуют файлы конфигурации CI/CD процессов, которые запускают команды для исполнения при определенном взаимодействии с репозиториями проектов, см. рис. 1. Автотесты могут быть специфичны для разных компонентов приложения, а вот автодеплой может запускаться простой командой docker-compose up -build с указанием контекста с SSH параметрами доступа к серверу.

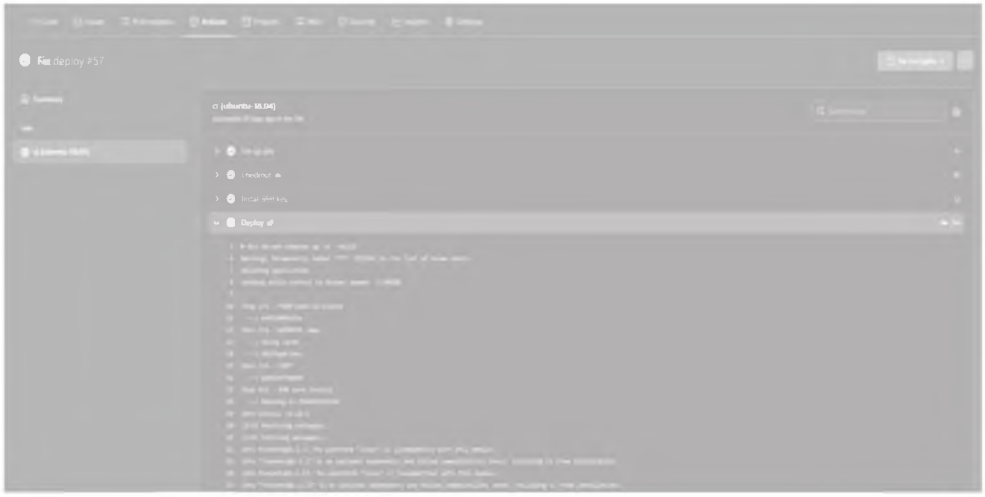

Рисунок 1 – Интерфейс GitHub Actions с примером запуска пайплайна деплоя

Но при наличии нескольких компонентов приложения невозможен одновременный запуск нескольких web-серверов для прослушивания 80 и 443 портов одновременно. А именно эти порты являются обязательными для клиент серверного общения по протоколам http и https. Решением является запуск отдельного web-сервера с функцией обратного прокси для приема всех запросов на сервер. С подобной задачей мог бы справиться Nginx, но в данном случае более правильным решением будет использование web-сервера Caddy. Он будет автоматически генерировать SSL(TLS) сертификаты безопасности, а также настройка всех компонентов приложения будет сводиться к простому перечню доменных имен и направления их проксирования. Так как корневой web-сервер не имеет привязки к конкретному компоненту приложения, более уместным будет его расположения в отдельном репозитории. Caddy также прост в контейнеризации, добавив Dockerfile и docker-compose.yml файлы, а также файлы конфигурации автодеплоя для обновления приложения могут быть также расположены внутри репозитория проекта.

По умолчанию все контейнеры, определенные в одном docker-compose.yml файле имеют доступ друг к другу по имени контейнера, что, например, позволяет Nginx напрямую обращаться к запущенному в другом контейнере конкретному компоненту приложения. Но когда необходима работа в связке компонентов приложения из разных репозиториев, необходимо работать в сети хоста, что является небезопасным, или создавать отдельную независимую сеть. Второй вариант просто выполняется простой командой docker create network server. Далее достаточно указать используемую сеть в docker-compose.yml файле каждого репозитория. Автоматизировать создание сети можно указав команду выше в файле конфигурации CI/CD репозитория корневого web-сервера.

В итоге на сервере будет реализована изолированная сеть server для созданного корневого web-сервера Caddy и всех подключенных к нему web-серверов отдельных компонентов приложений. А также множество небольших сетей для отдельных частей приложений – application network. Добавив к изображению примерную структуру самих контейнеров, можно получить следующую структуру компонентов внутри сервера, см. рис. 2.

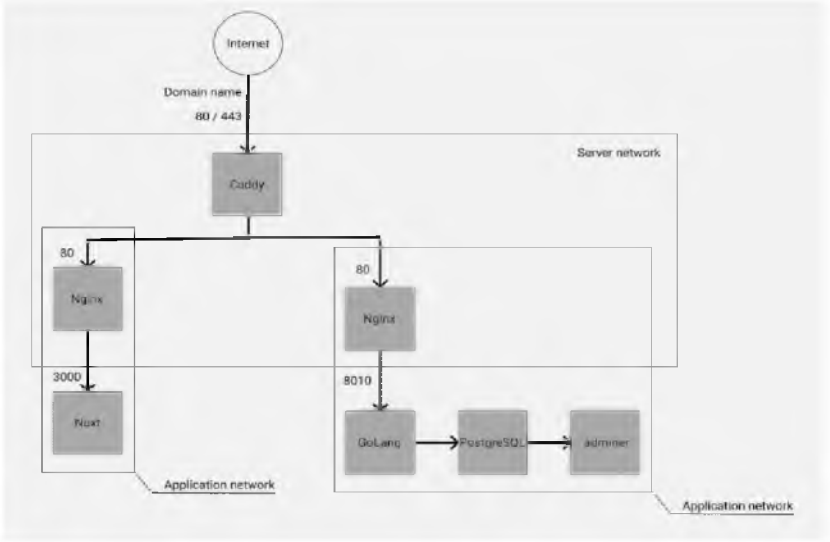

Рисунок 2 – Внутренняя структура компонентов сервера

Для добавления в создаваемый сервис возможности тестирования на отдельных версиях приложениятребуется решить проблемы генерации тестовых версий компонентов и удаления тестовых версий компонентов. Первое делается за счет указания настроек отдельного пайплайна с указанием уникального имени проекта для Docker и созданием уникального имени контейнера web-сервера отдельной части приложения. Далее из web-сервера Caddy можно находить нужные контейнеры по уникальному имени. Сам же Caddy будет понимать, к какому контейнеру осуществляется запрос, за счет уникальной структуры доменного имени. Например, основная версия сайта может быть доступна по непосредственному доменному имени (domain–name.com), а к тестовым версиям сайта по добавленному поддомену равному именованию ветки репозитория отдельной тестовой версии сайта (my-branch.domain-name.com). Для того, чтобы сайты на сервере были доступны по указанным доменным именам необходимо, чтобы доменное имя имело необходимые DNS записи. Caddy в рамках созданной server сети найдет нужный контейнер и передаст ему запрос для дальнейшей обработки.

Удаления тестовой ветки не является типовой задачей разработки и не существует готового алгоритма для решения этой проблемы. Реализация подобной задачи возможна на языке программирования Golang. Приложение будет запускаться в своем собственном контейнере, прослушивать открытый порт на сервере, принимать на него запросы от Webhook репозиториев и осуществлять процесс удаления тестовой версии сайта. Расположить исходный код программы удаления тестовой версии сайта возможно в репозитории корневого web-сервера Caddy.

Разработанное приложение удаления тестовых версий проекта будет принимать запросы от Webhook репозиториев, из полученной информации узнавать о том, с каким проектом и какой его версией требуется произвести операцию. Далее приложение подключается к своему хосту и может производить операции с контейнером. В частности, осуществлять удаление и иные операции с запущенными на сервере контейнерами. Необходимые параметры доступа к контексту сервера, особенно SSH ключ доступа необходимо скрывать от возможности чтения со стороны. При использовании GitHub можно использовать GitHub Secrets, а при использовании Gitlab использовать Environment переменные, заполняемые в настройках репозиториев. И в репозитории корневого web-сервера также возможно добавлять различные дополнительные инструменты, такие как, например, инструменты мониторинга работы приложений.

Подобную задачу выполняют ряд аналогичных сервисов и программ, однако все они имеют ряд недостатков для решения большинства задач.

Kubernetes – инструмент оркестрации контейнеров, выполняющий практически все вышеперечисленные задачи. Однако он требует запуск

корневого main узла для управления всеми запущенными подами. Его работа, в идеале, требует выделение отдельного сервера с большими затратами ресурсов. А также Kubernetes имеет высокий порог вхождения для большинства разработчиков, что делает его неподходящим для небольших и средних проектов [1,2].

Netlify – коммерческий сервис запуска тестовых версий приложений. Однако его использование является платным, а также он более подходит для тестовых версий проекта, но не для продуктовых.

Разработанная технология имеет больший спектр применения, меньшие финансовые и трудовые затраты и большие возможности для масштабирования и контроля [3].

Заключение. Приложение и архитектура сервера для решения, использующая разработанную технологию, позволяет избежать многие проблемы, с которыми сталкиваются разработчики ПО. Все компоненты приложения имеют необходимые настройки из репозиториев проектов, процессы автотестирования и обновления сервера полностью автоматизированы, соединение с приложением полностью безопасны, среда исполнения кода стабильна и легко переносима между серверами, а для проекта имеется возможность работы как продуктовой версии приложения, так и множества тестовых версий [5-6]. Дальнейшее масштабирование и развитие приложения просты и дешевы в реализации. Архитектура не завязана на конкретном проекте или компании, а значит ее применение возможно абсолютно для любых проектов. Также имеется возможность поэтапного перехода для проектов с наличием уже собственной устаревшей архитектуры.

Литература

1. Аверина, П. А. Контейнеризация понятным языком: от самых азов до тонкостей работы с Kubernetes / П. А. Аверина: сайт. – 2019 – URL: https://habr.com/ru/company/southbridge/blog/530226/ (дата обращения: 16.04.2022). – Текст: электронный.
2. О Kubernetes и серверной архитектуре: сайт. – 2022 – URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (дата обращения: 18.04.2022). – Текст: электронный.
3. Биденко, А.А. Образ приложения Asmy на Docker Hub / А.А. Биденко: сайт. – 2021 – URL: https://hub.docker.com/repository/docker/alexbidenko1998/admire-server-manager (дата обращения: 12.06.2022). – Текст: электронный.
4. Общие архитектуры веб-приложений: сайт. – 2021 – URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата обращения: 05.08.2021). – Текст: электронный.
5. Биденко, А. А. Официальный сайт приложения Asmy / А. А. Биденко: сайт. – 2022 – URL: https://asmy.pro/ (дата обращения: 12.06.2022). – Текст: электронный.
6. Биденко, А. А. DevOps или Как освободить многих разработчиков в один клик / А. А. Биденко: сайт. – 2018 – URL: https://tproger.ru/articles/devops-ili-kak-osvobodit-mnogih-razrabotchikov-v-odin-klik/ (дата обращения: 18.02.2021). – Текст: электронный.
7. Learn GitHub Actions: сайт. – 2020 – URL: https://docs.github.com/en/actions/learn-github-actions (дата обращения: 14.07.2021). – Текст: электронный.
8. Manage data in Docker: сайт. – 2020 – URL: https://docs.docker.com/storage/ (дата обращения: 30.01.2022). – Текст: электронный.
9. Network containers: сайт. – 2020 – URL: https://docs.docker.com/engine/tutorials/networkingcontainers/ (дата обращения: 18.12.2021). – Текст: электронный.
10. Web Authentication API: сайт. – 2019 – URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API (дата обращения: 13.03.2022). – Текст: электронный.
11. What is a web server: сайт. – 2019 – URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server (дата обращения: 11.03.2021). – Текст: электронный.
12. What is CI/CD: сайт. – 2021 – URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (дата обращения: 13.07.2021). – Текст: электронный.
13. What is Kubernetes: сайт. – 2020 – URL: https://www.redhat.com/en/topics/containers/what-is-kubernetes (дата обращения: 15.04.2022). – Текст: электронный.

References

1. Averina, P. A. Kontejnerizaciya ponyatnym yazykom: ot samyh azov do tonkostej raboty s Kubernetes / P. A. Averina: sajt. – 2019 – URL: https://habr.com/ru/company/southbridge/blog/530226/ (data obrashcheniya: 16.04.2022). – Tekst: elektronnyj.
2. O Kubernetes i servernoj arhitekture: sajt. – 2022 – URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (data obrashcheniya: 18.04.2022). – Tekst: elektronnyj.
3. Bidenko, A. A. Obraz prilozheniya Asmy na Docker Hub / A. A. Bidenko: sajt. – 2021 – URL: https://hub.docker.com/reposi-

tory/docker/alexbidenko1998/admire-server-manager (data obrashcheniya: 12.06.2022). – Tekst: elektronnyj.
4. Obshchie arhitektury veb-prilozhenij: sajt. – 2021 – URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (data obrashcheniya: 05.08.2021). – Tekst: elektronnyj.
5. Bidenko, A. A. Oficial'nyj sajt prilozheniya Asmy / A. A. Bidenko: sajt. – 2022 – URL: https://asmy.pro/ (data obrashcheniya: 12.06.2022). – Tekst: elektronnyj.
6. Bidenko, A. A. DevOps ili Kak osvobodit' mnogih razrabotchikov v odin klik / A. A. Bidenko: sajt. – 2018 – URL: https://tproger.ru/articles/devops-ili-kak-osvobodit-mnogih-razrabotchikov-v-odin-klik/ (data obrashcheniya: 18.02.2021). – Tekst: elektronnyj.
7. Learn GitHub Actions: sajt. – 2020 – URL: https://docs.github.com/en/actions/learn-github-actions (data obrashcheniya: 14.07.2021). – Tekst: elektronnyj.
8. Manage data in Docker: sajt. – 2020 – URL: https://docs.docker.com/storage/ (data obrashcheniya: 30.01.2022). – Tekst: elektronnyj.
9. Network containers: sajt. – 2020 – URL: https://docs.docker.com/engine/tutorials/networkingcontainers/ (data obrashcheniya: 18.12.2021). – Tekst: elektronnyj.
10. Web Authentication API: sajt. – 2019 – URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API (data obrashcheniya: 13.03.2022). – Tekst: elektronnyj.
11. What is a web server: sajt. – 2019 – URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server (data obrashcheniya: 11.03.2021). – Tekst: elektronnyj.
12. What is CI/CD: sajt. – 2021 – URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (data obrashcheniya: 13.07.2021). – Tekst: elektronnyj.
13. What is Kubernetes: sajt. – 2020 – URL: https://www.redhat.com/en/topics/containers/what-is-kubernetes (data obrashcheniya: 15.04.2022). – Tekst: elektronnyj.

УДК-37.035.6
DOI: 10.34046/aumsuomt105/40

АВТОМАТИЧЕСКОЕ ДИАГНОСТИРОВАНИЕ ПОДШИПНИКОВ ГРУЗОВЫХ НАСОСОВ С ИСПОЛЬЗОВАНИЕМ ВЕЙВЛЕТ-ПАКЕТНОГО ПРЕОБРАЗОВАНИЯ И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

И.К. Барсуков, аспирант

В статье приведены исследования, теоретические основы, материалы и методы диагностики посредством метода вейвлета. При пиковых нагрузках, переходных режимах или частой работы грузового насоса усиливается деградация компонентов насоса и особенно подшипников. Такие режимы (особенно первые два) работы увеличивает механическую вибрацию подшипников качения, что может привести к их преждевременному разрушению. Показания снимаются во время запуска насоса и фильтруются по среднему значению преобразуя в вейвлет-пакет. Отфильтрованные ряды используются для оценки взаимосвязи между максимальным смещением кривой и накопленными наработками. Оценочное уравнение, связанное со стандартами вибрации, устанавливает время пребывания в состояние деградации и производит моделирование процесса износа оборудования с учетом последовательности максимальных смещений кривых.

Ключевые слова: вейвлет, вейвлет-пакетное преобразование, грузовой насос, подшипник, моделирование, кривая.

AUTOMATIC DIAGNOSTICS OF CARGO PUMP BEARINGS USING WAVELET PACKET CONVERSION AND COMPUTER SIMULATION

I.K. Barsukov

The article presents research, theoretical foundations, materials and diagnostic methods using the wavelet method. At peak loads, transient modes or frequent operation of the cargo pump, the degradation of pump components and especially bearings increases. Such modes (especially the first two) of operation increase the mechanical vibration of rolling bearings, which can lead to their premature destruction. The readings are taken during the pump start and filtered by the average value, converting it into a wavelet packet. Filtered series are used to estimate the relationship between the maximum displacement of the curve and accumulated operating time. The evaluation equation associated with vibration standards sets the time spent in a state of degradation and simulates the process of equipment wear taking into account the sequence of maximum displacements of the curves.

Keywords: wavelet, wavelet-batch transformation, cargo pump, bearing, modeling, curve.