Устимова Снежана Сергеевна, магистрант, Финансовый университет при Правительстве Российской Федерации, г. Москва

ПРОБЛЕМА ОСУЩЕСТВЛЕНИЯ МОНИТОРИНГА НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ DEVOPS-ИНЖЕНЕРАМИ НА ПРЕДПРИЯТИЯХ СФЕРЫ ГОЗ

Аннотация. Постановка задачи: Профессия DevOps-инженеров с каждым годом становится все более востребованной. Они способствуют автоматизации процессов, улучшению коммуникации между командами и ускорению релизов программного обеспечения. Их деятельность направлена на повышение надежности и устойчивости IT-инфраструктуры, что позволяет компаниям быстрее адаптироваться к рыночным изменениям и внедрять инновационные решения. Однако предприятия, выполняющие гособоронзаказы (ГОЗ), сталкиваются с проблемой при внедрении некоторых практик DevOps, вызванной недостатком специализированного ПО, способного пройти сертификацию, доступного в ОС спец. назначения Astra Linux Special Edition (SE).

Цель: Изучить существующие инструменты, которые могут быть использованы в условиях закрытых госпредприятий, предложить альтернативный вариант в виде создания независимого и интегрируемого ПО.

Задачи исследования: Провести анализ современных трендов и способов осуществления мониторинга надежности в ИТ-предприятиях, выявить проблемы, связанные с их использованием в компаниях сферы ГОЗ, изучить стандарты, регламентирующие методики управления надежностью.

Результаты: Обозначена проблема, препятствующая реализации принципа «непрерывный мониторинг». Отмечено, что для реализации мониторинга имеется лишь одно доступное ПО, прошедшее сертификацию – Zabbix. Тем не менее, его использование не всегда представляется возможным ввиду 2 причин: необходимость наличия доступа к сети в компании и устаревании версии из репозитория. Предложен альтернативный вариант в виде создания собственного

интегрируемого инструмента для осуществления мониторинга надежности ПО на основании рекомендаций, изложенных в стандарте ГОСТ Р МЭК 62628-2021.
Практическая значимость: Полученные результаты могут быть использованы DevOps-инженерами, работающих в компаниях, разрабатывающих на Astra Linux SE. Рассмотренные модели также могут быть дополнены или изменены, исходя из целей мониторинга.

Abstract. Problem statement: The profession of DevOps engineers is becoming more and more in demand every year. They help automate processes, improve communication between teams, and accelerate software releases. Their activities are aimed at improving the reliability and sustainability of the IT infrastructure, which allows companies to quickly adapt to market changes and implement innovative solutions. However, enterprises that carry out state defense orders (GOZ) face a problem when implementing certain DevOps practices caused by a lack of specialized software capable of certification, available in the special-purpose Astra Linux Special Edition OS.

Purpose: Consider existing tools that can be used in closed state-owned enterprises, to propose an alternative option in the form of creating your own software.
Research tasks: Consider current trends and ways of reliability monitoring in IT enterprises, identify problems related to their use in companies in the field of public health, and study standards governing reliability management techniques.
Results: The problem hindering the implementation of the principle of "continuous monitoring" is identified. It is noted that there is only one certified software available for monitoring, Zabbix. However, its use is not always possible due to 2 reasons: the need for network access in the company and the outdated version from the repository. An alternative option is proposed in the form of creating its own integrated tool for monitoring software reliability based on the recommendations set out in the GOST R IEC 62628-2021 standard.
Practical significance: The results obtained can be used by DevOps engineers working in companies developing on Astra Linux SE. The models considered can also be supplemented or modified based on monitoring objectives.

Ключевые слова: DevOps, надежность ПО, мониторинг, Astra Linux SE, smодели жизненного цикла, модели безотказности ПО.

Keywords: DevOps, software reliability, monitoring, Astra Linux SE, lifecycle models, software reliability models.

ВВЕДЕНИЕ

В современном мире информационные технологии играют ключевую роль в развитии различных отраслей, и надежность программного обеспечения становится одним из главных факторов успешного функционирования IT-инфраструктуры. DevOps-инженеры, как специалисты, объединяющие разработку (Development) и эксплуатацию (Operations), играют важную роль в обеспечении непрерывной интеграции, доставки и деплоя приложений.

Одним из ключевых аспектов работы DevOps-инженера является мониторинг и управление надежностью программного обеспечения. Эффективная система мониторинга позволяет не только своевременно выявлять и устранять ошибки, но и прогнозировать потенциальные проблемы, обеспечивая стабильность и надежность работы приложений. Тем не менее, использование некоторых инструментов DevOps может быть ограничено политикой компании. В таких случаях приходится срочно искать альтернативные варианты либо разрабатывать собственные решения.

В статье выделены существующие подходы, принципы, практики и инструменты DevOps, а также предложены способы решения проблемы осуществлении мониторинга на государственных предприятиях.

1. Исследование методологии DevOps: принципы и практики. Взаимосвязь жизненного цикла ПО с деятельностью DevOps-инженеров

Заказы на изготовление программного обеспечения (ПО) на предприятиях обычно начинаются с тендера и формирования технического задания, в которых заказчик описывает свои требования. После заключения договора начинается старт жизненного цикла ПО.

Жизненный цикл представляет собой развитие системы, продукта, услуги, проекта или других изготовленных человеком объектов, начиная с этапа разработки концепции и заканчивая прекращением применения.

Стандарты, такие как ГОСТ Р ИСО/МЭК 12207-2010, описывают общую структуру этих процессов, не конкретизируя методы выполнения задач. Такой подход позволяет гибко подходить к реализации проектов в зависимости от специфики и требований заказчика.

В общем виде под жизненным циклом программного обеспечения (англ. Software Development Life Cycle, SDLC) принято понимать концепцию создания информационных систем (ИС), включающую их планирование, разработку, тестирование и развертывание. Концепция применима как к программным, так и к аппаратным, и к комбинированным ИС. Она помогает разработчикам создавать высококачественные продукты, соответствующие ожиданиям клиентов в установленные сроки и в рамках бюджета.

Принято выделять следующие этапы ЖЦ программного обеспечения:

1) планирование и анализ требований;
2) определение требований;
3) проектирование архитектуры;
4) разработка программного обеспечения;
5) тестирование продукта;
6) развертывание и сопровождение.

При разработке программного обеспечения очень важно сократить время, необходимое для производства продукта, соответствующего требованиям качества. Одним из главных ключей к тому, чтобы проект был готов вовремя, является хорошо отлаженный процесс его разработки на всех предложенных этапах [1].

Порядок следования вышеописанных этапов определяется по итогам выбора модели жизненного цикла ПО.

Однако чаще всего вне зависимости от выбора конкретной модели жизненного цикла ПО, будь то каскадная / V-модель / инкрементальная / RAD /

итеративная / спиральная или гибкая на практике оказывается, что команды все равно сталкиваются с различными сложностями при разработке и эксплуатации программного обеспечения, которые могут тормозить процесс и снижать качество конечного продукта [2]. Какие же это проблемы?

1) длительное время вывода продукта на рынок.

2) сложности интеграции и совместимости.

3) высокий уровень ошибок и дефектов ввиду ручного тестирования и недостаточного мониторинга.

4) недостаточная автоматизация и гибкость.

5) низкая коммуникация и сотрудничество между командами.

DevOps помогает решать вышеуказанные проблемы через интеграцию и автоматизацию процессов на всех этапах ЖЦ ПО.

DevOps можно также охарактеризовать фразой «всё непрерывно». Жизненный цикл DevOps можно разделить на 7 непрерывных этапов (7C), определяющих процесс разработки ПО от начала и до конца.

Рассмотрим подробнее 7C жизненного цикла DevOps [3].

1. Continuous Development. На этом этапе собираются и обсуждаются с заинтересованными сторонами требования к проекту. Кроме того, на основе отзывов клиентов формируется список задач, который разбивается на более мелкие выпуски и этапы для непрерывной разработки ПО. После согласования бизнес-требований команда разработчиков приступает к написанию кода. Это непрерывный процесс, в ходе которого разработчики должны вносить изменения в код при любых изменениях в требованиях к проекту. Используются инструменты контроля версий: git, mercurial, subversion.

2. Continuous Integration. На этом этапе обновлённый код или дополнительные функции и возможности разрабатываются и интегрируются в существующий код. Кроме того, на этом этапе на каждом шаге с помощью модульного тестирования выявляются и устраняются ошибки в коде, а затем исходный код соответствующим образом изменяется.

3. Continuous Testing. На этом этапе аналитики качества непрерывно тестируют ПО на наличие ошибок и проблем с помощью контейнеров Docker. В случае обнаружения ошибки код отправляется обратно на этап интеграции для внесения изменений. Автоматизированное тестирование также сокращает время и усилия, необходимые для получения качественных результатов. На этом этапе команды используют такие инструменты, как Selenium.

4. Continuous Deployment. На данном этапе окончательный код развертывается на производственных серверах. Непрерывное развертывание включает в себя управление конфигурацией для обеспечения точного и бесперебойного развертывания кода на серверах. Команды разработчиков развертывают код на серверах и планируют обновления для серверов, поддерживая согласованность конфигураций на протяжении всего производственного процесса.

5. Continuous Feedback. Непрерывная обратная связь в DevOps – это непрерывный процесс сбора, анализа и обработки обратной связи на протяжении всего ЖЦ разработки программного обеспечения.

6. Continuous Monitoring. На этом этапе функциональность и возможности приложения постоянно отслеживаются для выявления системных ошибок, таких как нехватка памяти, недоступность сервера и т. д. Этот процесс помогает ИТ-команде быстро выявлять проблемы, связанные с производительностью приложения, и первопричину, которая их вызывает. Если ИТ-команда обнаруживает какую-либо критическую проблему, приложение снова проходит через весь цикл DevOps, чтобы найти решение.

7. Continuous Operations. CO автоматизирует процесс запуска приложения и его обновлений. Для устранения простоев используются системы управления контейнерами и оркестраторы, такие как Docker и Kubernetes.

Таким образом, процесс разработки программного обеспечения в рамках DevOps – это непрерывная деятельность, охватывающая все этапы жизненного цикла программного обеспечения – от этапа планирования до сопровождения и мониторинга. За счет интеграции и автоматизации всех этих фаз команды могут

более оперативно предоставлять качественные продукты и быстро адаптироваться к изменениям и возникающим проблемам.

2. Гипотеза и методика исследования

Основная идея статьи – исследование возможности внедрения концепции непрерывного мониторинга в закрытые государственные предприятия, которые ограничены в использовании популярных и современных технологий DevOps.

Объектом исследования является вся совокупность IT-инфраструктуры и DevOps-окружения. В роли предмета исследования выступают методы и инструменты мониторинга и показатели надежности программного обеспечения.

Целью работы является анализ существующих инструментов, которые могут быть использованы в условиях закрытых госпредприятий, а также предложение альтернативных вариантов в виде создания независимого и интегрируемого ПО.

Методика исследования включает в себя следующие этапы:

1. Анализ рынка инструментов мониторинга в закрытых системах;
2. Выявление недостатков при использовании существующего софта;
3. Обзор альтернативного способа мониторинга.

3. Анализ существующих методик мониторинга надежноти предприятиях сферы ГОЗ

Необходимым условием для создания качественного, стабильного и безопасного продукта, который удовлетворит потребности пользователей и обеспечит конкурентное преимущество на рынке, является обеспечение надежности программного обеспечения на всех этапах его жизненного цикла – от планирования до поддержки [4].

Особую важность надежность ПО представляет для государственных заказчиков. Надежность – одно из важнейших требований к разрабатываемой системе. Как правило, госзаказчики начинают запрашивать информацию о

надежности ПО еще на этапе разработки программы, так как стремятся получить представление об эффективности текущих процессов и зрелости продукта [5].

Надежность программного обеспечения на территории Российской Федерации регламентирует стандарт ГОСТ Р МЭК 62626-2021.

Согласно этому стандарту, надежность программного обеспечения (англ. software dependability) определяется как способность программного блока работать в составе системы в соответствии с установленными требованиями. Система не удовлетворяет требованию надежности при возникновении неисправности.

Неисправность программного обеспечения, ошибка (англ. software fault, bug) – это такое состояние объекта ПО, которое препятствует его работе в соответствии с установленными требованиями.

Проявление неисправности программного обеспечения называется отказом ПО (англ. software failure).

Обеспечение надежности должно быть неотъемлемой частью планов проекта и входить в перечень задач при разработке системы, охватывая все этапы.

В дополнение к стандарту ГОСТ Р МЭК 62626-2021 задействовать руководство по проектированию надежных систем, изложенное в стандарте МЭК 60300-3-15.

Для обеспечения надежности программного обеспечения рекомендуется выполнение следующих процедур:

а) определение целей применения ПО и требований, относящихся к его жизненному циклу и условиям его применения;

б) определение применимых показателей надежности ПО, относящихся к проекту программного обеспечения;

в) анализ адекватности процессов управления надежностью и наличия ресурсов для поддержки разработки проекта и изготовления ПО;

г) установление требований к ПО и целей обеспечения надежности;

д) классификация неисправностей ПО и определение соответствующих параметров программного обеспечения (для реализации стратегии обеспечения надежности программного обеспечения;

е) применение соответствующих методов надежности при проектировании и изготовлении ПО;

ж) инициирование повышения надежности, если применимо, с учетом различных ограничений при адаптации проекта;

з) мониторинг процессов разработки и изготовления для управления и получения обратной связи, необходимых для поддержания работоспособности ПО и обеспечения надежности системы в эксплуатации.

Последний пункт вызывает сложности, когда речь заходит о государственных заказчиках. В частности, когда мы имеем дело с системами, находящимися в компетенции ФСБ России (Системы Администрации Президента, Совета Безопасности, Федерального Собрания, Правительства, Конституционного Суда Российской Федерации, ФСО России и ФСБ России).

Единственной операционной системой, которая соответствует требованиям безопасности всех регуляторов к ПО для импортозамещения, является ОС Astra Linux Special Edition. Она предназначена для автоматизированных систем в защищенном исполнении, обрабатывающих информацию со степенью секретности «совершенно секретно» включительно. На данный момент доступны версии 1.7 и 1.8.

Согласно требованию 8 Центра ФСБ России разработка и тестирование программного обеспечения в таких системах может происходить исключительно с использованием тех средств, которые поставляются вместе с установочным диском данной ОС.

Единственное ПО с назначением мониторинга, поддерживаемое Astra Linux – это Zabbix.

Zabbix – программное решение с открытым исходным кодом для мониторинга и управления ИТ-инфраструктурой, написанная Алексеем

Владышевым. Оно обладает широким спектром возможностей, среди которых можно выделить следующие:

1) мониторинг производительности серверов и сетевого оборудования: позволяет отслеживать загрузку процессоров, использование памяти, дискового пространства и сетевых интерфейсов в реальном времени;

2) отслеживание состояния сервисов и приложений – система может собирать данные о доступности, времени ответа и производительности различных сервисов и приложений;

3) визуализация данных – Zabbix предоставляет возможность построения графиков и дашбордов для наглядного представления состояния ИТ-инфраструктуры;

4) уведомления и алерты – система может автоматически уведомлять администраторов о критических событиях или отклонениях от нормы;

5) анализ трендов и прогнозирование – Zabbix помогает выявлять тенденции и предсказывать возможные проблемы на основе исторических данных;

6) Журналирование и аудит событий – система ведет логи всех операций и событий для последующего анализа и отчетности.

7) Интеграция с другими системами – Zabbix может взаимодействовать с другими инструментами и системами для более полного и комплексного мониторинга.

Эти возможности делают Zabbix мощным инструментом для обеспечения надежности и безопасности ИТ-инфраструктур.

Несмотря на богатый функционал, сразу можно выделить 2 проблемы, связанные с мониторингом ПО в условиях реализации государственного оборонного заказа (ГОЗ):

1. Zabbix, как единственное ПО, требует подключения репозиториев установочного диска, диска обновления и диска разработчика. В доступной инструкции по обновлению указана необходимость подключения к сети. И это является первой проблемой для большинства предприятий, работающих по

ГОЗу, поскольку в них запрещено использование всемирной паутины на рабочих компьютерах.

2. Если предположить, что в компании все же разрешено подключение к интернету, то возникает вторая проблема, связанная с устареванием версии Zabbix из пакетов репозитория Astra Linux. При последней версии 7.0 установленная из пакета версия Zabbix будет 4.0.

Указанные проблемы мотивируют на поиск альтернативного подхода к мониторингу надежности программных систем. Требуется разработать систему мониторинга, которая не только сможет оценивать основные показатели производительности программных компонентов и пр., но и должна быть интегрирована в непрерывный DevOps-процесс.

Предлагаемым в рамках данной статьи решение – это разработка собственного ПО, которая для вычисления показателей, связанных с надежностью, будет использовать модели безотказности системы.

Модели безотказности ПО – это инструменты и методы, используемые для оценки и прогнозирования надежности компьютерных программ. Они помогают определить, насколько вероятно, что программа будет работать без сбоев в различных условиях эксплуатации.

На сегодняшний день существует множество моделей и критериев для оценки вероятности безотказной работы и оценки характеристик ПО. Наибольшее количество моделей разработано для удовлетворения конкретных потребностей в течение жизненного цикла программного обеспечения. Примеры включают модель прогнозирования в процессе проектирования программного обеспечения и модель оценки дополнительного времени тестирования, необходимого перед выпуском программного обеспечения. Некоторые модели разработаны для прогнозирования надежности программного обеспечения еще до написания кода. Ввод данных для прогнозирования надежности в таком случае часто основывается на данных работы и применения аналогичной программной системы. Другие модели используют для оценки тенденций

повышения интеграции системы программного обеспечения, на основе промежуточных тестовых входных данных.

Перечислим некоторые наименования моделей, рекомендованных стандартом:

- Musa-basic;
- Musa-Okumoto;
- Jelinski-Moranda;
- Littlewood-Verrall;
- Schneidewind;
- Geometric;
- Brooks-Motley;
- Bayesian;
- Keene.

Не существует единой модели, способной охватить весь жизненный цикл программного обеспечения. На практике для определения вероятности безотказной работы программного обеспечения часто используют несколько моделей. Для выбора модели часто используют статистические методы, такие как критерии согласия, чтобы проверить, насколько хорошо модель соответствует набору наблюдений. Большая часть моделей для оценки вероятности безотказной работы программного обеспечения автоматизированы из-за их итерационных вычислений. Интерпретация результатов моделирования безотказности требует практического опыта и знаний в области надежности.

Заключение

В процессе исследования были обнаружены барьеры для реализации DevOps практик в организациях с чувствительной политикой использования средств разработки. Был предложен способ решений возникающей проблемы, а именно – разработка внутренней система мониторинга, которая для оценки состояния надежности ПО будет использовать данные различных моделей

безотказности ПО. Дальнейшие исследования будут сосредоточены на внедрении создаваемой системы в непрерывной конвейер DevOps.

Литература:

1. Волошин М. Описание жизненного цикла программного обеспечения // Журнал «Интернаука».2022. № 7-4 (230). С. 5-10.
2. Хермаван А., Маник Л. П. Влияние внедрения DevOps на качество командной работы при разработке программного обеспечения // Журнал инженерии информационных систем и бизнес-аналитики. 2021. Т. 7. №. 1 С. 84.
3. Жизненный цикл DevOps: 7 этапов, подробно описанных на примерах. // Симформ : [сайт]. 2023.URL: https://www.simform.com/blog/devops-lifecycle/ (дата обращения: 14.01.2025).
4. Бертолино А., Гульельмо Де Анджелис, Геррьеро А., Миранда Б., Пьетрантуоно Р. DevOpRET: Непрерывное тестирование надежности в DevOps // Журнал программного обеспечения: эволюция и процессы. 2020. Т. 35.
5. Уильямс Бейтс М.Л., Энрике И. Овьедо. Надежность программного обеспечения в среде непрерывной интеграции DevOps // Ежегодный симпозиум по надежности и сопровождаемости (RAMS). 2021. С. 1-4.

Literature

1. Voloshyn M. Software architecture description lifecycle // Журнал «Интернаука». 2022. No 7-4 (230). P. 5-10.
2. Hermawan A., Manik L. P. The effect of DevOps implementation on teamwork quality in software development // Journal of Information Systems Engineering and Business Intelligence. 2021. Vol. 7. No. 1. P. 84.
3. DevOps Lifecycle: 7 phases explained in detail with examples.Текст : электронный // Simform : [website]. 2023. URL: https://www.simform.com/blog/devops-lifecycle/ (date of request: 14.01.2025).

4. Bertolino A., Guglielmo De Angelis, Guerriero A., Miranda B., Pietrantuono R. DevOpRET: Continuous reliability testing in DevOps // Journal of Software: Evolution and Process. 2020. Vol. 35.

5. Williams Bates M.L., Enrique I. Oviedo. Software reliability in a DevOps continuous integration environment // Annual reliability and maintainability symposium (RAMS). 2021. P. 1-4.