Задание №2 - Описание жизненного цикла задачи (разработки нового функционала)

Здравствуйте!

Методология работы в команде - Scrum или Kanban или Waterfall.

Мой идеальный вариант жизненного цикла задачи такой:

1. В проекте появилась UserStory, ее поставил PM.
2. Макет дизайна уже готов и согласован заказчиком, изменения дизайна возможно будут, но в адекватных пределах.
3. DevTeam ознакомилась с user-story, по мере необходимости создает подзадачи уже самостоятельно.
4. DevTeam разрабатывает функционал.
5. DevTeam проводит код-ревью нового функционала.
6. DevTeam покрывает функционал unit-тестами.
7. DevTeam покрывает функционал сквозными-тестами(cypress). P.S.: для frontend.
8. Задача выкладывается на staging-environment из feature-ветки.
9. Прогоняются тесты.
10. QATeam берут задачу в работу и проводят ручное тестирование до тех пор пока функционал не будет соответствовать требованиям качества QATeam идет цикл его доработки и тестирования.
11. QATeam пишут интеграционные тесты.
12. Задача мержится в dev-ветку и уходит на staging-environment.
13. Происходит показ клиенту.
14. Прогоняются тесты.
15. QATeam берут задачу в работу и проводят тестирование в dev-ветке.
16. PM показывает результат заказчику.
17. Задача мержится в master-ветку и уходит на production-environment.

Задачи DevOps-инженера в данном цикле разработке:

1. Настройка инфраструктуры: staging-environment, production-environment.
2. Настройка pipeline(автоматизация развертывания изменений).
3. Настройка систем мониторинга.

4. Контроль здоровья приложений.
5. Реагирование на ошибки.

ПРО ИНФРАСТРУКТУРУ

Инфраструктура проекта будет на Kubernetes + cloud UI для Kubernetes.
Поскольку это K8S - все проекты будут Docker'изированы. Образы и их версии будут храниться в хранилище облака.

Облако предоставляет возможность максимально просто и быстро через UI обновлять образы.

Сам по себе K8S позволяет работать с zero down-time access обновлением образов.

ПРО КОНТРОЛЬ ВЕРСИЙ

Код будет храниться в git на github/gitlab.

Методология работы с системой контроля версий - git-flow. Это подразумевает что будут следующие ветки:
1. master - для production environment
2. dev - для staging environment
3. feature - для development

ПРО PIPELINE

Pipeline будет сконфигурирован так чтобы автоматически собирать новые образы из кода и пушить их в репозиторий образов с новым тегом. Тег образа будет определять его версию.

staging/production стенды будут обновляться при получении образов приложений новых версий.
Триггер на staging - при слиянии в ветку в dev
Триггер на production - при слиянии в ветку master

ПРО DEVELOPMENT-environment

Типовой интернет-магазины будет иметь в себе как-минимум

следующие сервисы: backend-container, frontend-container, database-container.

При разработке не всегда с фичей идут изменения во все сервисы.

Для того чтобы тестировать фичи из feature-веток можно создать несколько дополнительных изолированных тестовых стендов и в них изменять необходимые образы вручную и тестировать.

ПРО ЛОГИРОВАНИЕ

Для меня идеально было бы чтобы все ошибки nginx(и от приложений которые выведены куда-то отдельно, например, sentry или обработанные в try/catch ошибки) валились в отдельный канал в slack или в telegram.

При возникновении определенной одной и той же ошибки более 100 раз, чтобы срабатывал триггер и отправлял мне на email сообщение с деталями, системно повторяющий баг.

В идеальном мире ошибок не должно быть на production, но такого не бывает. Такой контроль поможет увеличить uptime и надежность приложения.

ПРО ОСТАЛЬНОЕ
Terraform, Ansible, Prometheus, Sonarqube.