Спринт 1

ЗАДАЧА

Опишите инфраструктуру будущего проекта в виде кода с инструкциями по развертке, нужен кластер *Kubernetes* и служебный сервер (будем называть его *srv*).

1

Выбираем облачный провайдер и инфраструктуру.

В качестве облака подойдет Яндекс.Облако

Нам нужно три сервера:

- два сервера в одном кластере *Kubernetes*: *1 master* и *1 app*;
- сервер *srv* для инструментов мониторинга, логгирования и сборок контейнеров.

2

Описываем инфраструктуру.

Описывать инфраструктуру мы будем, конечно, в *Terraform*.

Совет: лучше создать под наши конфигурации отдельную группу проектов в *Git*, например, *devops.*

Пишем в *README.md* инструкцию по развертке конфигураций в облаке. Никаких секретов в коде быть не должно.

3

Автоматизируем установку.

Надо реализовать возможность установки на сервер всех необходимых нам настроек и пакетов, будь то *docker-compose*, *gitlab-runner* или наши публичные ключи для доступа по *SSH*. Положите код автоматизации в *Git*-репозиторий.

Результат должен быть такой, чтобы после запуска подобной автоматизации на сервере устанавливалось почти всё, что нужно.

Совсем полностью исключать ручные действия не надо, но в таком случае их надо описать в том же *README.md* и положить в репозиторий.

Совет: лучше использовать для этого *Ansible.*

Спринт 2

В прошлом спринте мы с вами описали и создали нашу будущую инфраструктуру.

Теперь давайте её используем — соберем и задеплоим приложение из нашего *Git* в созданный кластер *Kubernetes*. Разделим всё на шаги.

1

Клонируем репозиторий, **собираем его на сервере** *srv.*

Исходники простого приложения можно взять здесь. Это простое приложение на *Django* с уже написанным *Dockerfile*. Приложение работает с *PostgreSQL*, в самом репозитории уже есть реализация *docker-compose* — её можно брать за референс при написании *Helm*-чарта.

Необходимо склонировать репозиторий выше к себе в *Git* и настроить пайплайн с этапом сборки образа и отправки его в любой *docker registry*. Для пайплайнов можно использовать *GitLab*, *Jenkins* или *GitHub Actions* — кому что нравится.
Рекомендуем *GitLab*.

2

Описываем приложение в *Helm*-**чарт**.

Описываем приложение в виде конфигов в *Helm*-чарте. По сути, там только два контейнера — с базой и приложением, так что ничего сложного в этом нет. Стоит хранить данные в БД с помощью *PVC* в *Kubernetes*.

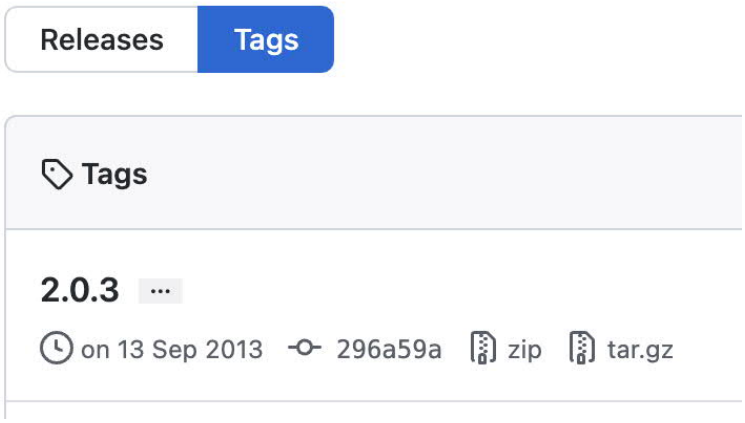
3

Описываем стадию деплоя в *Helm.*

Настраиваем деплой стадию пайплайна. Применяем *Helm*-чарт в наш кластер. Нужно сделать так, чтобы наше приложение разворачивалось после сборки в *Kubernetes* и было доступно по бесплатному домену или на *IP*-адресе с выбранным портом.

Для деплоя должен использоваться свежесобранный образ. По возможности нужно реализовать сборку из тегов в *Git*, где тег репозитория в *Git* будет равен тегу собираемого образа.

Например:



Чтобы создание такого тега запускало пайплайн на сборку образа с таким именем *hub.docker.com/skillfactory/testapp:2.0.3*.

А на этом самый важный этап подходит к концу. А теперь сладкое под конец — в следующем спринте нам нужно настроить мониторинг и сборку логов приложения.

В следующем спринте мы с вами создадим комфортные условия для работы нашей команды — добавим возможность смотреть логи и отслеживать метрики. Это точно всем пригодится!

Спринт 3

Мы с вами задеплоили наше приложение в созданную инфраструктуру.

Теперь — вишенка на торте — настройка мониторинга и логирования. Давайте декомпозируем.

1

Настройка сборки логов.

Представьте, что вы разработчик, и вам нужно оперативно получать информацию с ошибками работы приложения.

Выберите инструмент, с помощью которого такой функционал можно предоставить. Нужно собирать логи работы пода приложения. Хранить это всё можно либо в самом кластере *Kubernetes*, либо на *srv*-сервере.

2

Выбор метрик для мониторинга.

Так, теперь творческий этап. Допустим, наше приложение имеет для нас некоторую важность. Мы бы хотели знать, когда пользователь не может на него попасть — время отклика, сертификат, статус код и так далее. Выберите метрики и инструмент, с помощью которого будем отслеживать его состояние.

Также мы хотели бы знать, когда место на *srv*-сервере подходит к концу.

Важно! Весь мониторинг должен находиться на *srv*-сервере, чтобы в случае падения кластера мы все равно могли узнать об этом.

3

Настройка дашборда.

Ко всему прочему хотелось бы и наблюдать за метриками в разрезе времени. Для этого мы можем использовать *Grafana* и *Zabbix* — что больше понравилось.

4

Алертинг.

А теперь добавим уведомления в наш любимый мессенджер, точнее в ваш любимый мессенджер. Обязательно протестируйте отправку уведомлений. Попробуйте «убить»

приложение самостоятельно, и засеките время от инцидента до получения уведомления. Если время адекватное, то можно считать, что вы справились с этим проектом!

Самое время положить все возможные конфигурации в *Git*-репозиторий, если вы этого ещё не сделали.