УДК 004.02

АНАЛИЗ ПОДХОДОВ АВТОМАТИЗАЦИИ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ, ДОСТАВКИ, ПРОВЕРКИ КАЧЕСТВА И МОНИТОРИНГА ВЕБ-ПРИЛОЖЕНИЯ

А. Д. Гончаренко

Белорусский государственный университет,
пр. Независимости, 4, 220030, г. Минск, Беларусь, fpm.goncharenkoAD@bsu.by
Научный руководитель: Давидовская М.И., старший преподаватель

Объектом исследования является веб-приложение на основе среды разработки Flask, библиотек ElasticSearch, SQLAlchemy, API для работы с текстом статей. Для развертывания создаваемого приложений была реализована инфраструктура, автоматизированы процессы по доставке кода, тестирования. Исследованы подходы реализации макета с использованием средств описания инфраструктуры как кода, построения процессов непрерывной интеграции на основе GitLab CI и непрерывного развертывания с применением ArgoCD, средств автоматизации и проектирования схемы ветвления в репозитории системы контроля версий git.

Ключевые слова: Python; Flask; Terraform; Kubernetes; GitLab CI; ArgoCD.

Проектирование облачной инфраструктуры стало неотъемлемой частью современного информационного мира. В условиях постоянных изменений и увеличивающейся сложности требований к инфраструктуре использование современных инструментов и методов становится критически важным для эффективной работы и развития бизнес-процессов.

Одним из таких инструментов, которые позволяют управлять и автоматизировать облачные ресурсы, является Terraform/Terragrunt. Это инструменты, которые позволяют создавать, настраивать и управлять облачными ресурсами с помощью декларативного описания. Применение системы непрерывной интеграции Terraform и систем непрерывной доставки ArgoCD позволяет значительно упростить и ускорить процесс создания и управления облачной инфраструктурой.

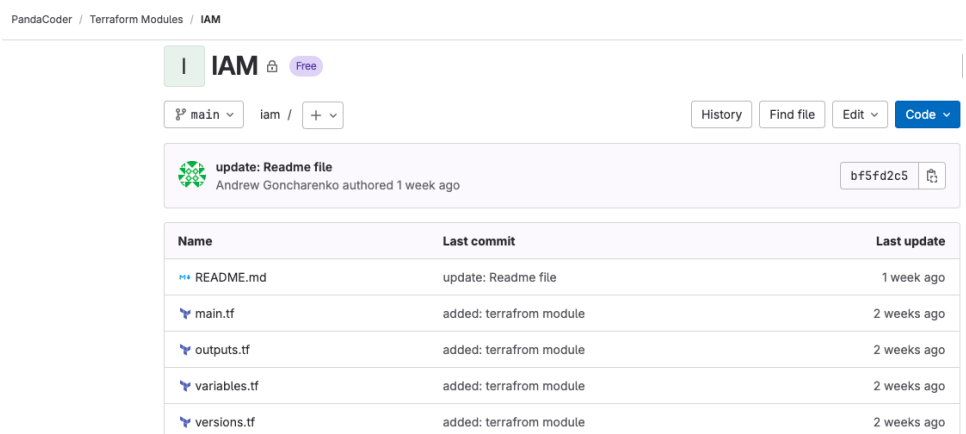

Рис. 1. Репозиторий Terraform модуля

Описание инфраструктуры как кода

Terraform – это один из инструментов инфраструктуры как кода (Infrastructure as Code – IaC) [1], используемый командами DevOps для автоматизации различных инфраструктурных задач. Используется в связке с Terragrunt. Terragrunt – это инструмент IaC, который служит для управления конфигурацией Terraform. Он предоставляет средства для организации и управления конфигурацией Terraform, такие как управление переменными, модулирование

конфигурации и обработка зависимостей между конфигурациями Terraform. Terragrunt упрощает процесс работы с Terraform, делая его более модульным, удобным.

Процесс описания инфраструктуры состоит из нескольких этапов:

5. Написание Terraform-модуля (VPC, IAM, K8S, …).
6. Публикация модуля в Terraform Module Regstry.
7. Переиспользование модуля в Terragrunt с описанием конфигурации (передача значения переменной, описанной в модуле).

terragrunt.hcl 1.22 KiB Blame Edit Replace Delete

```
terraform {
  source = "tfr://gitlab.com/pandacoder/yc/iam?version=1.0.0"
}

locals {
  environment = read_terragrunt_config(find_in_parent_folders("environment.hcl"))
}

include "root" {
  path = find_in_parent_folders()
}

dependency "dev-cf" {
  config_path = "../../folder"
  mock_outputs = {
    folder_id = "dummy_id"
  }

  mock_outputs_allowed_terraform_commands = ["init", "validate", "plan"]
}

inputs = {
  folder_id = dependency.dev-cf.outputs.folder_id

  name                = local.environment.locals.dev-cf.iam.service_accounts.yac-wh-sa-dev-cr.name
  description         = local.environment.locals.dev-cf.iam.service_accounts.yac-wh-sa-dev-cr.description
  roles               = local.environment.locals.dev-cf.iam.service_accounts.yac-wh-sa-dev-cr.roles
  create_auth_key     = local.environment.locals.dev-cf.iam.service_accounts.yac-wh-sa-dev-cr.create_auth_key
  auth_key_description = local.environment.locals.dev-cf.iam.service_accounts.yac-wh-sa-dev-cr.auth_key_description

}
```

Рис. 2. Использование Terraform-модуля в Terragrunt

Структура и архитектура приложения

Работа веб-приложения начинается с анализа конфигурационного файла __init__.py в котором прописываются основные настройки проекта, подключается база данных и инициализируются параметры, генерируется ключ безопасности и описывается вызов основных элементов приложения: models.py, routes.py, parsing.py.

Действия авторизованного пользователя по открытию статей, переходы между страницами сайта обрабатываются кодом из файла маршрутизации запросов routes.py и и файла models.py. Продемонстрируем работу routes.py. Каждое действие по получению и отправке данных (методы GET и POST) отслеживает при помощи @app.route в который передаем страницу перехода и связанные с ней методы. Рассмотрим запрос на просмотр статьи.

```
@app.route('/posts/<int:id>', methods=['GET'])
def posts_detail(id):
    article = Article.query.get(id)
    blob_reader = io.BytesIO(article.ImageID)
    base64_content = base64.b64encode(blob_reader.read()).decode('utf-8')
    img_src = f"{base64_content}"

    return render_template("posts_detail.html", article=article, image_data=img_src)
```

Рис. 3. Код открытия статьи

В @app.route определяем, что при переходе /posts/<int:id> страница будет поддерживать только GET метод, который будет возвращать пользователю текст статьи. Через <int:id> определяется, какая именно статья должна быть открыта. Далее в методе получаем по id статью и уже возвращаем ее html страницу.

Одним из пунктов разработки микросервисной архитектуры является принцип, который говорит о том, что при остановке работы одного сервиса остальные продолжат свою работу корректно. Следовательно, для сбора и систематизации статей (парсинга) и последующей обработки будет разрабатываться отдельное приложение на Python. Определим схему работы такого сервиса:

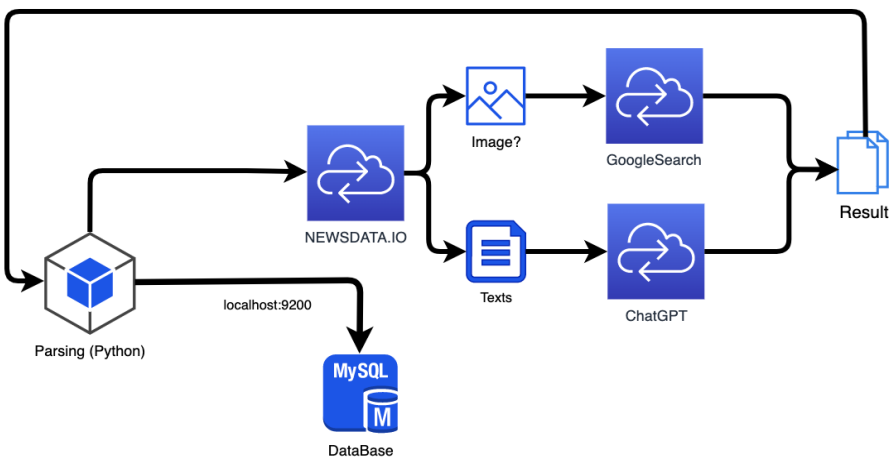

Рис. 4. Локальное представление работы сервиса

Стратегия ветвления

В модели GitFlow есть одна основная ветвь разработки со строгим доступом к ней. Ее часто называют веткой develop, подчеркивания размещение разрабатываемой версии приложения. Стабильная версия приложения публикуется в ветке master, которая используется для актуальной версии продукта.

Чтобы обеспечить совместную работу над текущей рабочей версией приложения, разработчики создают ветки с префиксом feature из ветки develop. По окончании работы они создают запросы на внесение изменений (pull request). pull request обеспечивают возможность комментирования и обсуждения реализации. После согласования запрос на изменение принимается и сливается с основной веткой.

Как только основная ветка пригодна для выпуска, создается отдельная ветка для подготовки финальной версии. Приложение тестируется и готовится к публикации для конечных пользователей. Завершается выпуск версии приложения публикацией в ветку master и созданием метки версии продукта.

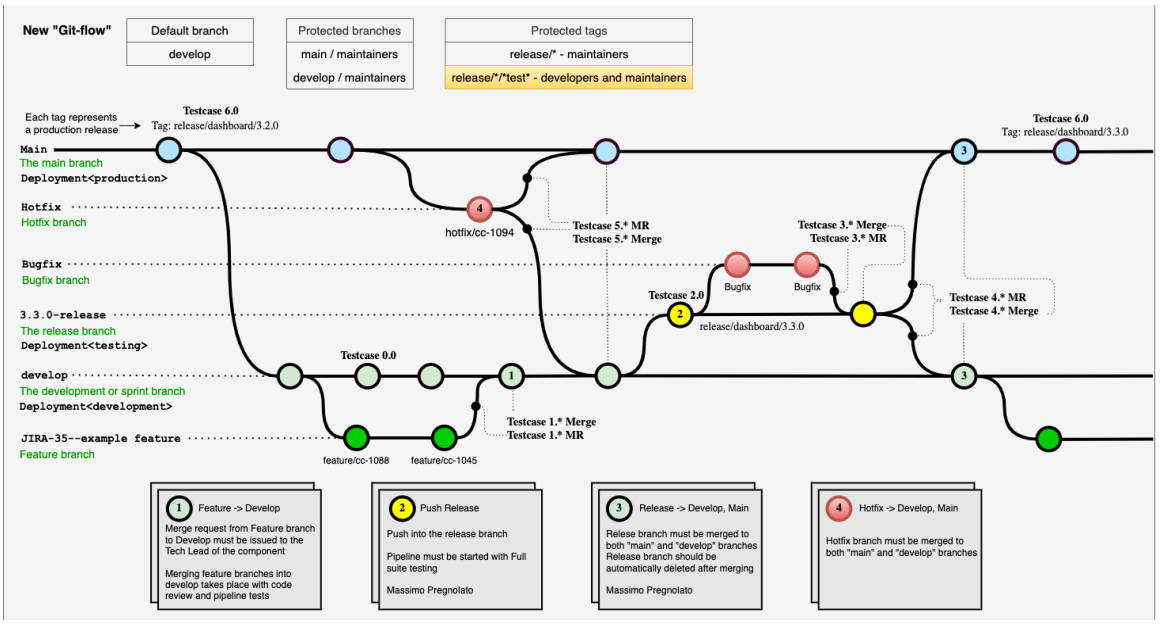

Рис. 5. Стратегия ветвления

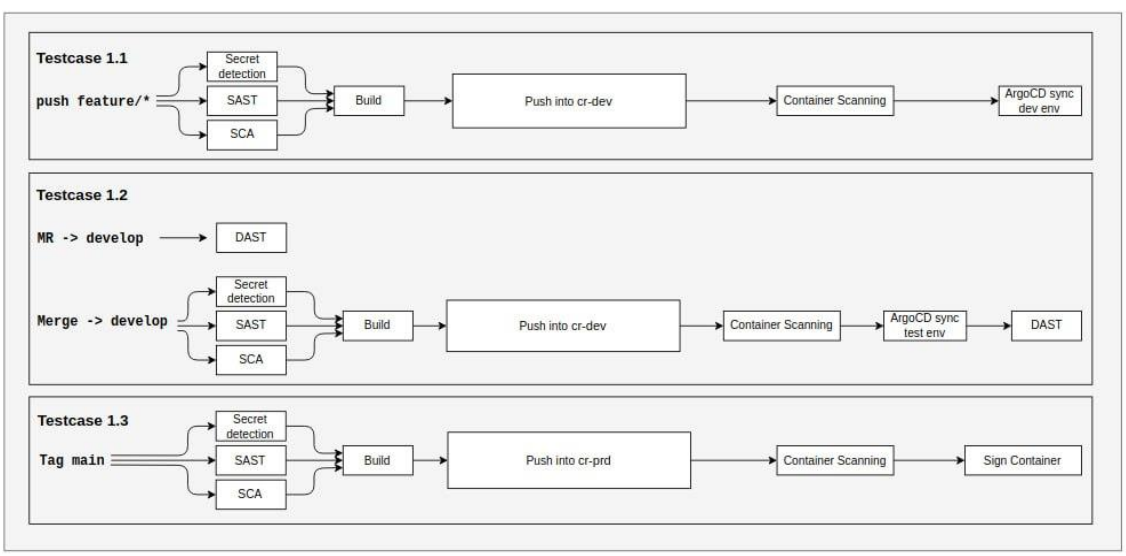

Рис. 6. Проверка безопасности решения

Организация CD процесса с использованием ArgoCD

ArgoCD – это инструмент, предназначенный для непрерывной доставки и управления приложениями в Kubernetes. Использование ArgoCD представляет собой ценное решение для автоматизации процесса доставки и управления приложениями. Он обеспечивает декларативное управление конфигурациями, непрерывную доставку, управление множеством окружений, визуализацию конфигураций и состояний, делая процесс управления приложениями более эффективным, простым и надежным.

Дальнейший процесс развертывания приложения заключается в написание helm инструкций, на основании которых ArgoCD будет создавать наименьшие развертываемые вычислительные единицы с контейнерами приложения. Для helm-конфигураций создан отдельный репозиторий.

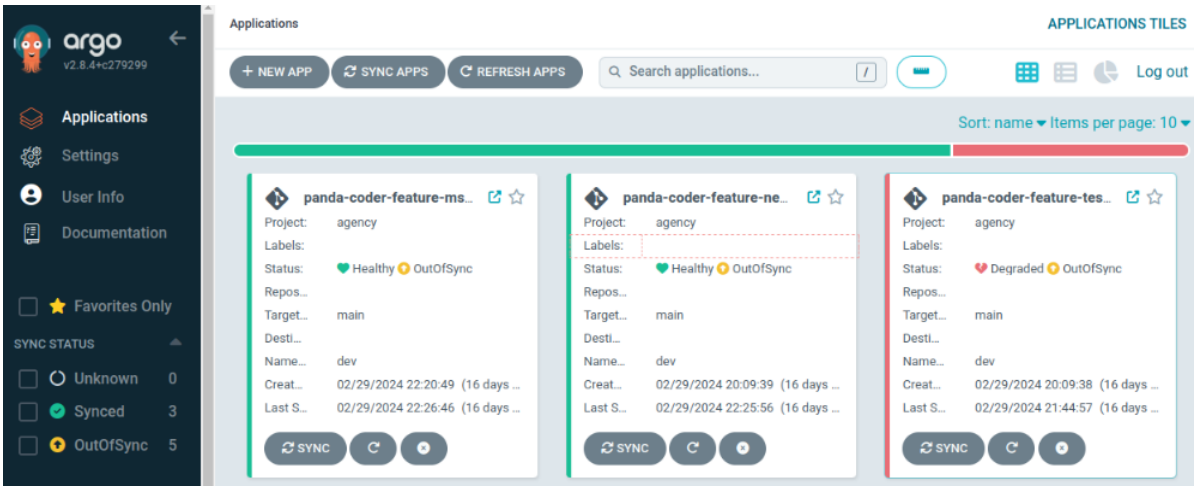

Рис. 7. ArgoCD и проекты в различных NS

Библиографические ссылки

1. Брикман Е. Terraform. Инфраструктура на уровне кода 2-е издание: учеб. пособие. СПб: “Питер”, 2020. С. 28-156.

2. Гончаренко А. Д., Давидовская М. И. Проектирование веб-приложения на основе Flask и Python // Научные исследования XXI века. 2023. № 1. С. 40-41.
URL: http://scientific-research.ru/files/JOURNAL--1--21-.pdf.

3. Haider R. Web API development with Python // Руководство для начинающих по использованию Flask и FastAPI: учеб. пособие. CloudBytes, 2021. С. 14-123.

4. Grinberg M. The New And Improved Flask Mega-Tutorial // Углубленное изучение Flask: учеб. пособие / Independently published, 2018. С. 158-215.