Andersen DevOps course exam

Marchenko Ivan

Presentation plan:

- Applications
- CI/CD
- Chosen Workflows
- Workflows implementation
- Infrastructure implementation
- Terraform template
- Ideal CI/CD pipeline
- Demonstration

Applications

Python app:

- Flask framework.
- Gunicorn WSGI server.
- Image size: 8.88 MB.
- GET & POST requests.
- 8080 port exposed.
- Workflow used: Git flow
- Message: Hello World 1

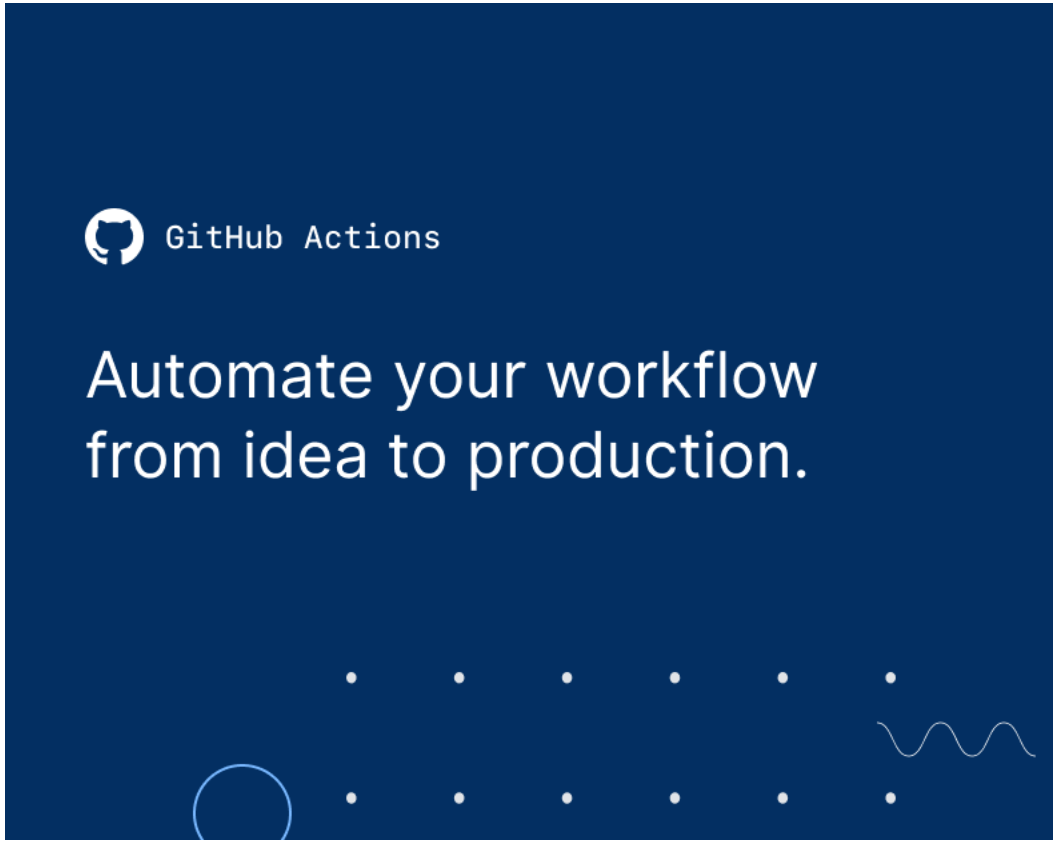
Golang app:

- Gin framework.
- -
- Image size: 9.06 MB.
- GET & POST requests.
- 8080 port exposed.
- Workflow used: Github flow
- Message: Hello World 2

CI/CD

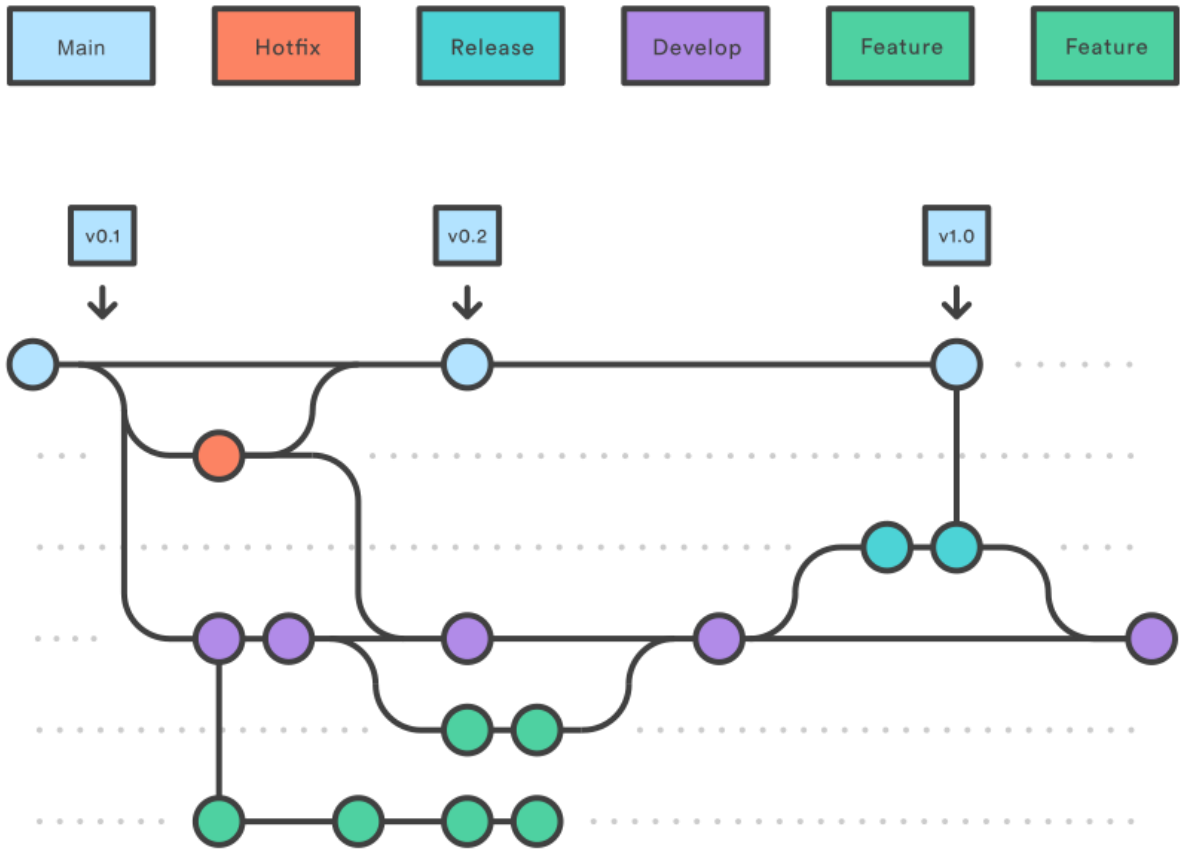
For my exam task I've chosen the Github Actions CI/CD platform because of these benefits:

- Integration with the Github and big amount of triggers for Github events.
- There is no need to use third-party CI/CD platforms when everything can be done in Github.
- Plenty amount of Actions: from simple Telegram notifications to Terraform integration.
- Easy-to-use YALM format code.
- There is no need to run a dedicated server for CI/CD platform.



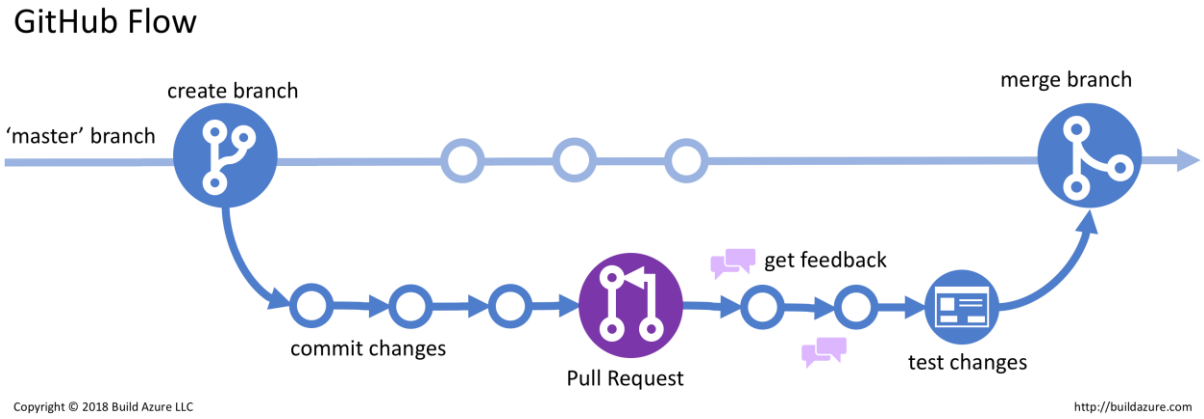
Chosen workflows | Gitflow

- Gitflow workflow uses several primary branches instead of one main branch.
- Every developer is working in his feature branches until feature is done. Developers can’t interact with the main branch.
- Project history is stored in several branches which includes Hotfix, Release, Develop and Main branch itself.
- There is a high risk of merge conflicts and Gitflow has fallen in popularity in favor of trunk-based workflows.

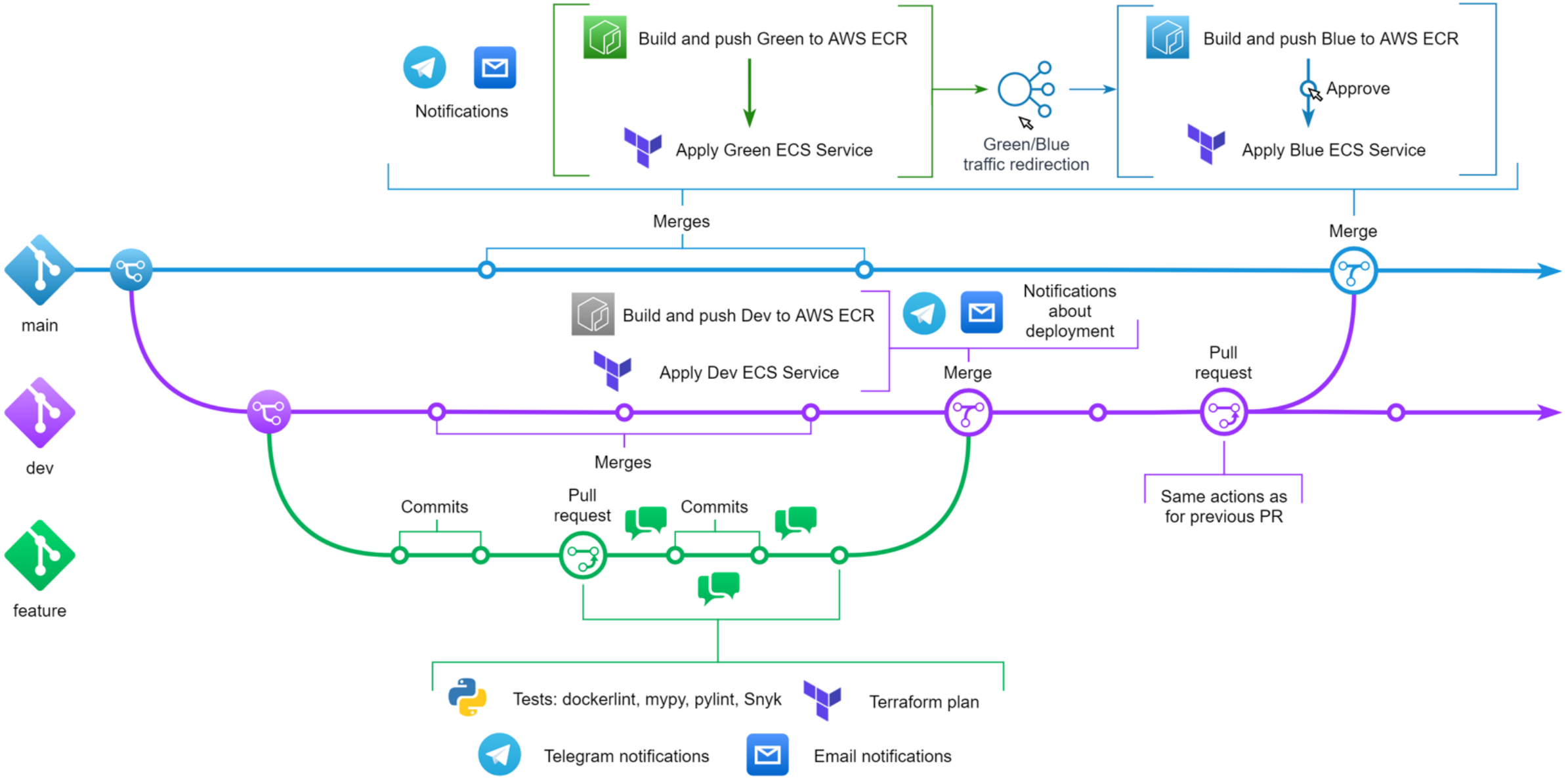


Chosen workflows | Github flow

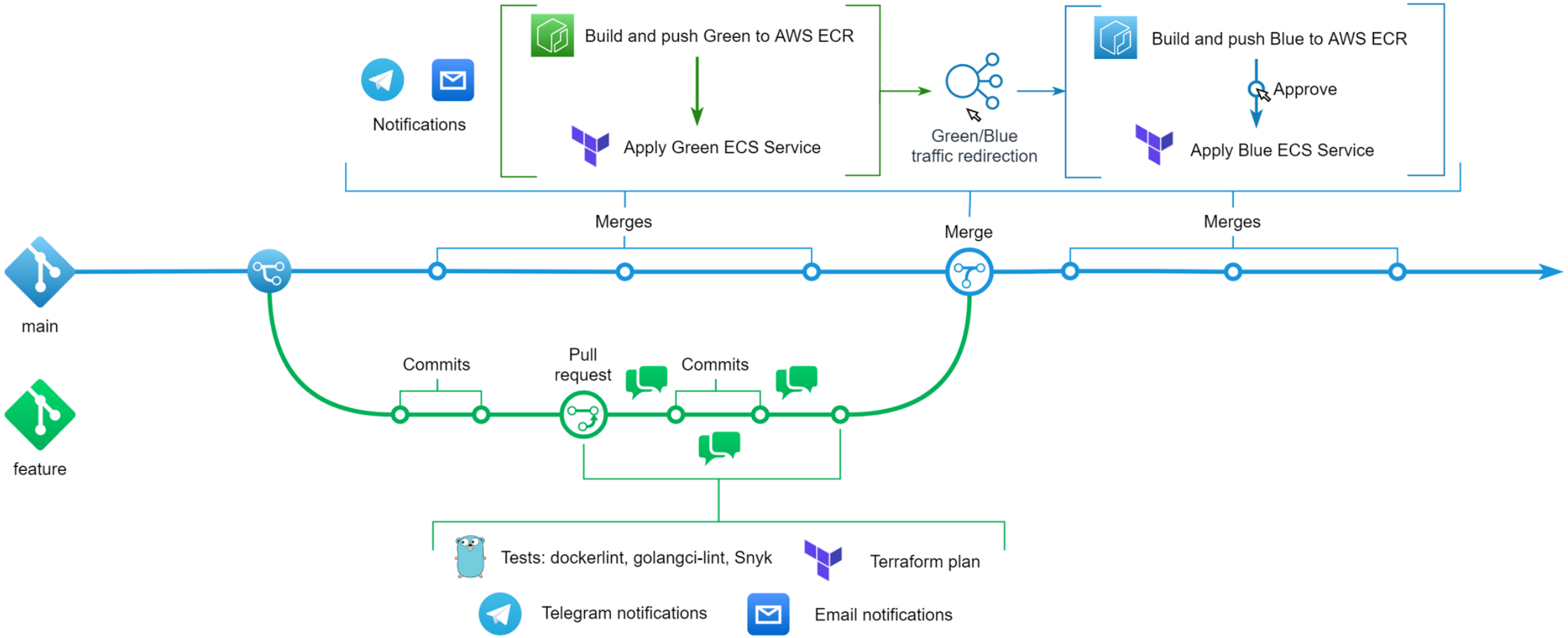
- Github flow is a simple workflow with only one main branch.
- Every developer is working in his feature branch and able to create a pull request to the main branch.
- After changes are merged to the main code is going for tests, staging and production.
- Today the Github flow is a most popular workflow because of microservices popularity that require for high commit rate.



Workflows implementation | Gitflow

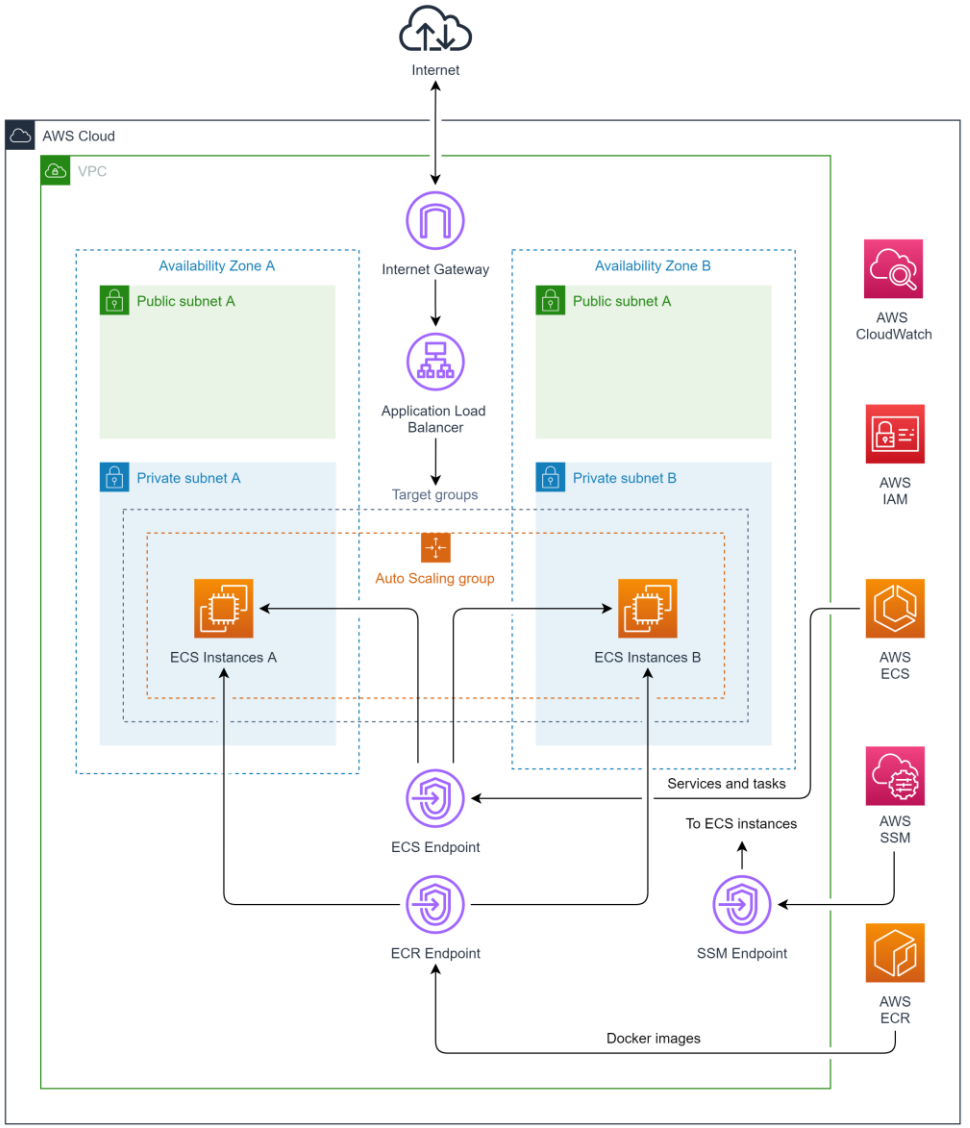


Workflows implementation | Github flow



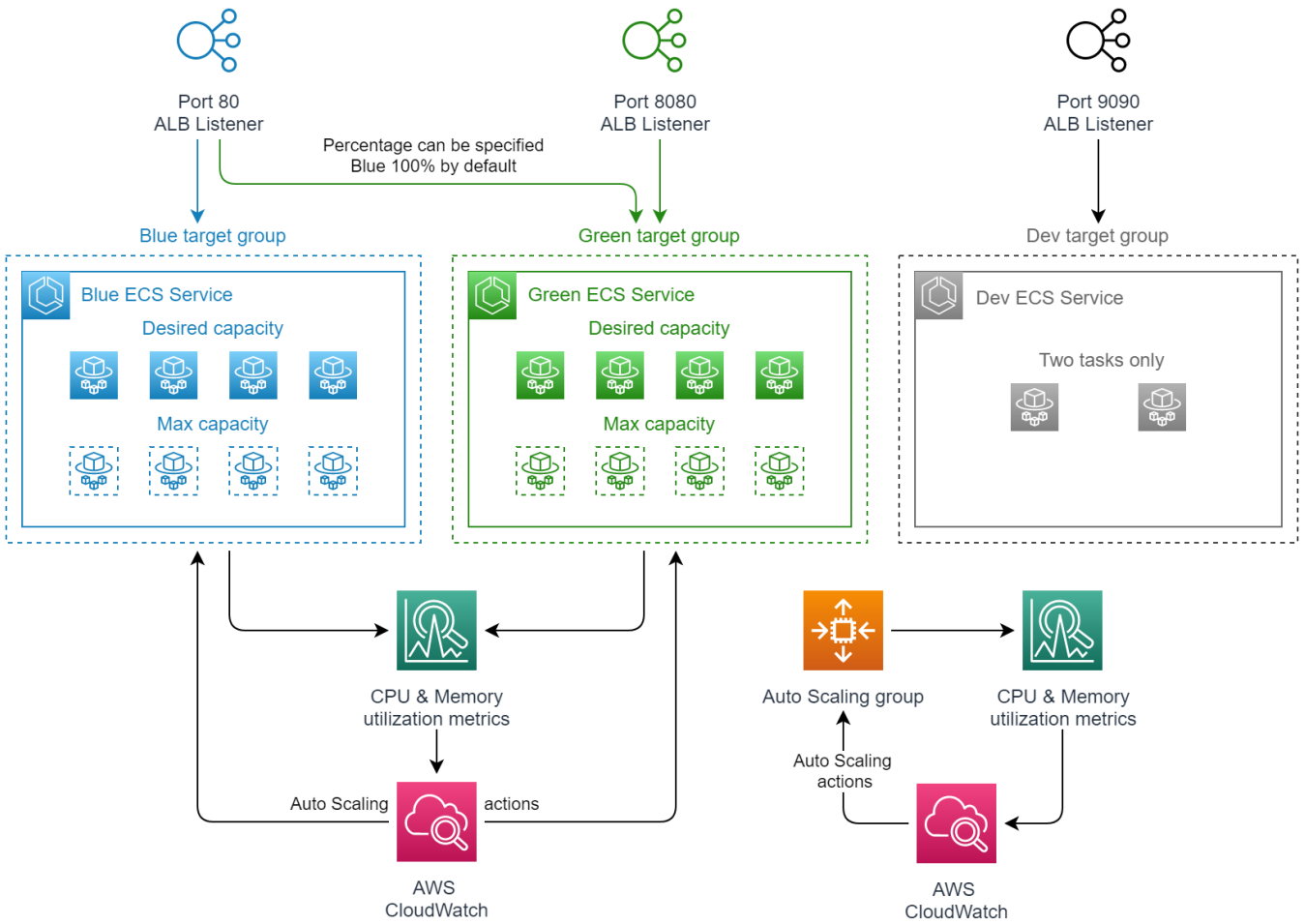
Infrastructure implementation

- AWS ECS Cluster based on AWS EC2 Auto Scaling group.
- Application Load Balancer that is balancing load between Services Tasks.
- Ability to change traffic between Green and Blue services.
- AWS Services Endpoints for ECS Instances in private subnets. Endpoints allows to control ECS Tasks & Services and upload images from ECR Repositories
- ECS Instances are in private subnets and there is no bastion host. How to connect to them? Simple: via SSM from AWS Console or AWS CLI.



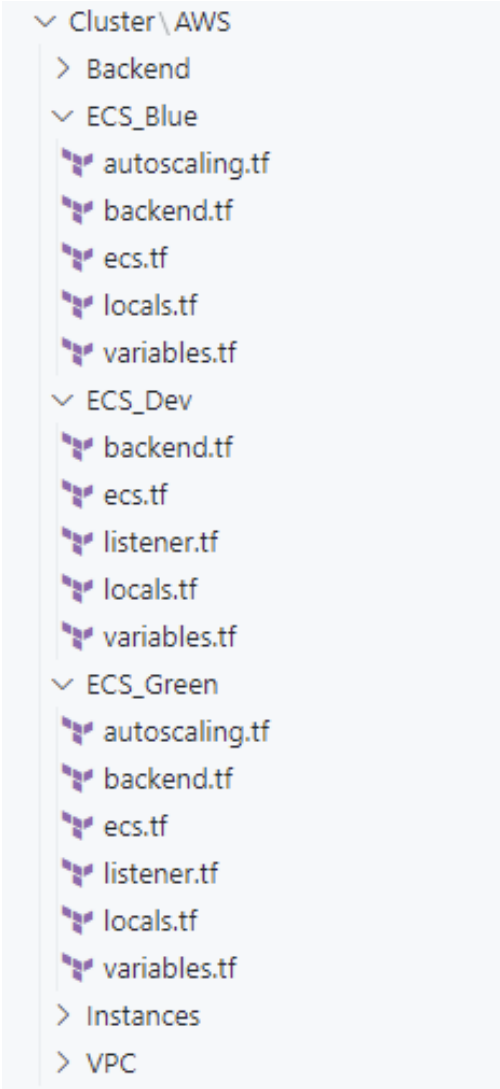
Infrastructure implementation

- Traffic between Blue and Green services can be specified by one of the five presets: blue, blue-80, split, green-80 and green.
- There is the Github Actions workflow called “Redirect Traffic” which allows to do it without interactions with CLI.
- Every ECS Service have different listeners. If you want to test Green or Dev, then connect to the ALB via 8080 or 9090 port.
- AWS CloudWatch alarms allows to monitor CPU & memory usage of the services and take Auto Scaling actions if one of the metrics is >80% usage.



Terraform template

- Terraform template consists of several directories created for every resource. It allows to create, change or destroy resources separately.
- Each created resource have its own remote state in the S3 Bucket with DynamoDB locking.
- In implemented workflows each resource is created automatically in Github Actions workflows.
- Creation or destruction of the resources can be triggered automatically by PRs & pushes or manually with workflow dispatch.



Ideal CI/CD pipeline

There are some features I would like to add to my CI/CD pipelines and infrastructure so they will be closer to the ideal:

CI pipeline:

- Unit tests.
- Integration tests.
- More integration with Github: comments, labels, tags, etc.

And for **CD pipeline** too:

- QA.
- Logging.
- Auto tests.
- Monitoring.

- And of course there is no ideal CI/CD universal pipelines.
- Everything is depends on situation, tasks and type of project and service.

For now there are some things missed in the **AWS Infrastructure**:

- SSL/TLS secure connection.
- Domain name.
- Metrics for ECS Development service.
- Secrets management.
- Variables for resources in Terraform template are messed up and need to be reworked.

Demonstration