How 50 years old company migrates to K8S

Freedom as a service

About us

Yavor Petkov

Senior Software Engineer at Software AG

Interests: AI and Formula 1

Pavel Malinov

Cloud Ops/Automation at Software AG

Interests: Beer && Gaming |

Content

1. Introduction
2. Previous setup
3. Current setup
4. CI/CD stages
5. What went wrong
6. Future
7. Q & A

Introduction

50 years of experience

Cloud

Kubernetes

Production Readiness Checklist

Started with Gruntwork* checklist for going live. We are not in great shape.

Where we started:

- Lack of monitoring coverage and log collection (too many alarms, too few logs collected and harvested)
- No automation, no CI/CD (no CI at any stage - all manual deployments via bash scripts, CD ???)
- No automated tests... and still no
- Low cost-optimization (not even AWS RI)

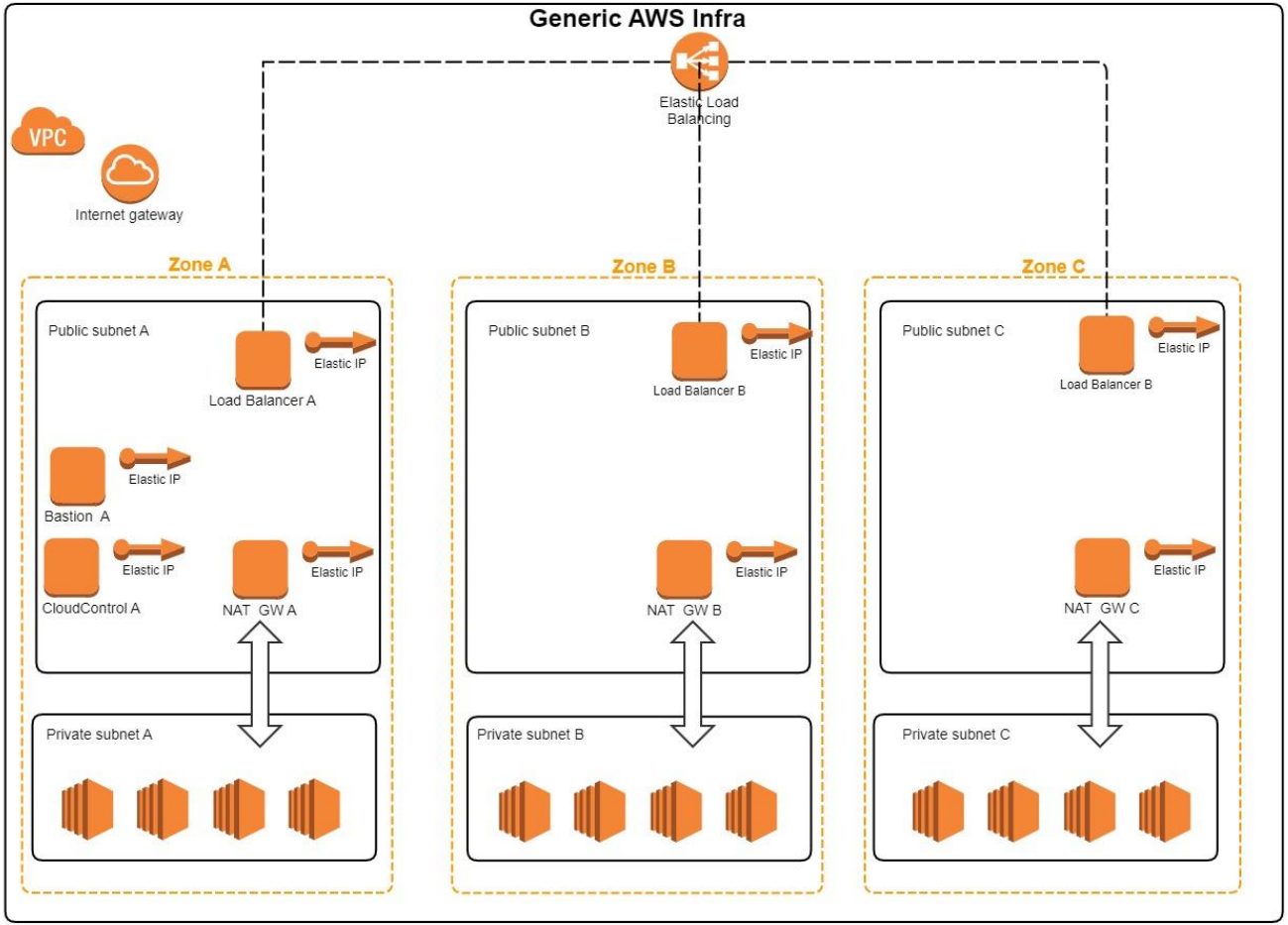
* https://www.gruntwork.io/devops-checklist/

Previous Setup & Cloud

Cloud Foundry (CNCF Project)

AWS infra

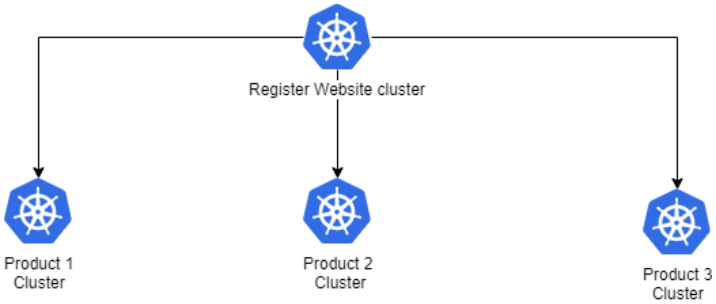
- Nothing WoW
- Did the Job



Current Setup

Per product clusters (EKS)

- AWS ALB Ingress Controller
- External DNS
- EFS – provisioner



Amazon EKS

argo

Public Clouds

CI/CD Stages

Develop

Staging

Pre-prod

Prod

Develop

- Build and Deploy - HELM + Skaffold + Jenkins

- Ingress

- Infrastructure

Staging

Build and Deploy - HELM + Skaffold + Jenkins

Ingress

AWS ALB Ingress Controller

Infrastructure

Pre/Prod

- Bitbucket pipelines and terraform modules
- Private cluster waiting on feature in the roadmap*
- Encryption via the new options in AWS for EC2
- Logging Control Plane to AWS CloudWatch
- ArgoCD bootstrapped in the cluster. One per cluster. Played with kustomize but did not fit our use case
- UI of ArgoCD is the main selling point but it is HTTP2 and grpc dependant: no ALB in AWS

*https://github.com/aws/containers-roadmap/issues/108

Monitoring & Logging

- Prometheus
  - *No Thanos as of now.*
- EFK centralized stack
  - AWS Elasticsearch, Kibana, FluentD

Fluentd is a swiss army knife with a bazooka attached. Great to work with it, can do everything, easy to break and lose hope in humanity.

What went wrong

- Automation
- Build small components
- Do the migration gradually
- Build environments as close as possible
- Only terraform or No terraform
- Bitbucket pipelines
- Strict pod limits and affinity needs to be in place to ensure some stability

Automation

Problem: We deleted our deployment

Solution: User access

Problem: Connection to CDN

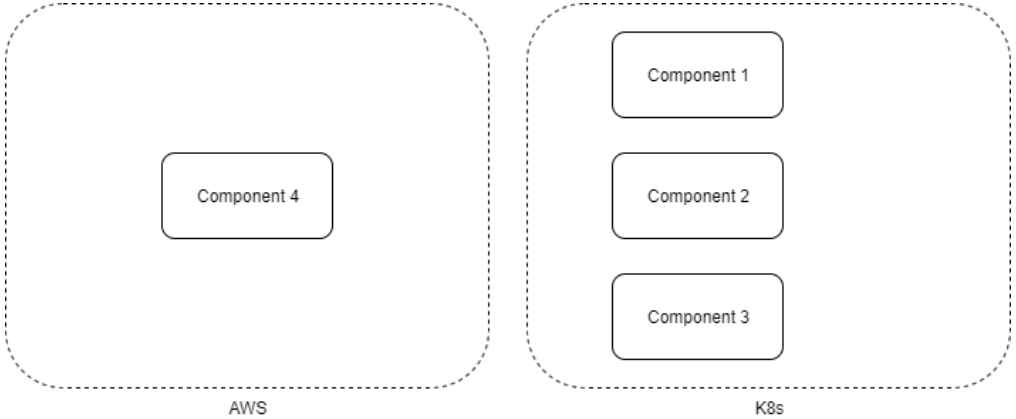
Solution: Automation

Build small components

Build small components

Do the migration gradually

- Migrate 4 components to K8s
- Leave 1 on the old setup
- No connectivity with the old setup



Build environments as close as possible

- Ingress differences
- Volume Providers
- Deployment Tools

TF + k8s provided resources

Once LB controller creates needed resources you cannot use terraform delete. We need to work on it.

```
module.aws_eks_controlplane.aws_eks_cluster.eks: Still destroying... (ID: promtest3-small-SC-9793, 8m50s elapsed)
module.vpc.aws_internet_gateway.this: Still destroying... (ID: igw-01d170c5af3dbca16, 9m30s elapsed)
module.vpc.aws_subnet.public.0: Still destroying... (ID: subnet-0e89f39c7d77301a1, 9m30s elapsed)
module.vpc.aws_subnet.public.2: Still destroying... (ID: subnet-0b52b1d027a874e39, 9m30s elapsed)
module.aws_eks_controlplane.aws_eks_cluster.eks: Still destroying... (ID: promtest3-small-SC-9793, 9m0s elapsed)
```

BitBucket Pipelines

- Our workflow has outgrown the current capabilities of BitBucket Pipelines
  - *Looking into : Gitlab, AWS Pipelines, CircleCI*
- Bitbucket has basic features
  - *A lot of "hacks" are needed to make it work*

Future

- Provide standard environments on every stage
- Automate migration of application from source to containers
- Bitbucket -> Gitlab?
- Templating CD - self bootstrapping workflow


software AG
Freedom as a Service