A Cloud-Based DevOps Toolchain for Efficient Software Development

Ruiyang Ding

Master's Thesis in Computer Science
EIT Digital Master's Programme in Cloud Computing and Services

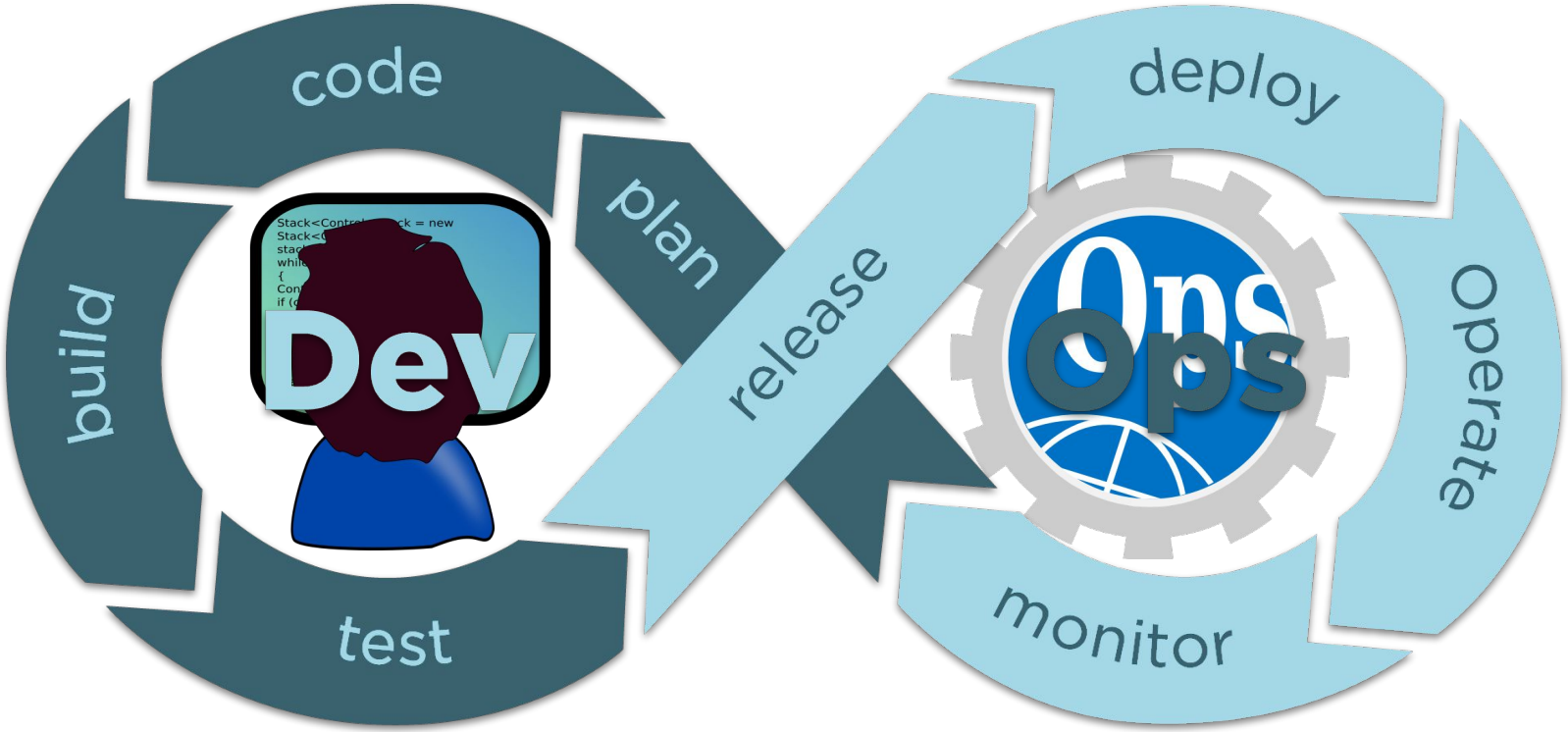
Supervisor: Prof.dr.ir. D.H.J. Epema
Company Supervisor: Mikko Drocan

31.08.2020

About Eficode

- Leading DevOps company in Europe
- Helping the customer in their DevOps transformation.

Introduction - DevOps



DevOps - Components

Culture

Automation

Organization

Monitoring & Measurement

DevOps Toolchain

"The integration between tools that specialised in different aspects of the DevOps ecosystem, which support and coordinate the DevOps practices."

Serverless

"A cloud execution model in which
the cloud provider manages the server and resources allocation"

Compared with Infrastructure as a service (IaaS):

- Event Driven
  IaaS: Keep running
- Managed Resources Allocation
  Iaas: Server creation and configurations are needed
- Pay-per-use, only pay for the code execution time
  IaaS: Pay for type of VM and the rental time of this VM

Serverless in AWS

eficode × aws

Advanced AWS Partner

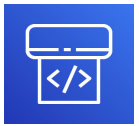
> We want to see how AWS serverless platform can helps in building the DevOps toolchains

DevOps 🧡 AWS Serverless ?

Computing

Monitoring

DevOps



.....

AWS serverless platform

TU Delft

Problem 1:

There is not enough research on how serverless can be used in DevOps toolchain

😀 DevOps for Serverless Application Development?

1. DevOps for Serverless Applications: Design, deploy, and monitor your serverless applications using DevOps practices (Shashikant Bangera, 2018)

2. Implementation of a DevOps pipeline for serverless applications (Ivanov, Vitalii and Smolander, Kari, 2018)

😕 Use Serverless in DevOps?

1. Several blogs such as “The Serverless Path to DevOps” (Sharjeel Yusuf, Feb 27, 2020)

2. No directly related paper so far

DevOps 🧡 Serverless ?

Problem 2

Now the software teams have two kinds of toolchains to select.

- Non-integrated DevOps Toolchain:
  Individual tools which were stand-alone and from different vendors

- Integrated Toolchain:
  Delivered as an single platform from an single vendor

AWS
DevOps
Tools

Research Questions

1. How can serverless computing services in Amazon Web Services enhance the DevOps toolchain?

2. How does the integrated toolchain build with AWS DevOps Tools compare with the traditional non-integrated toolchain?

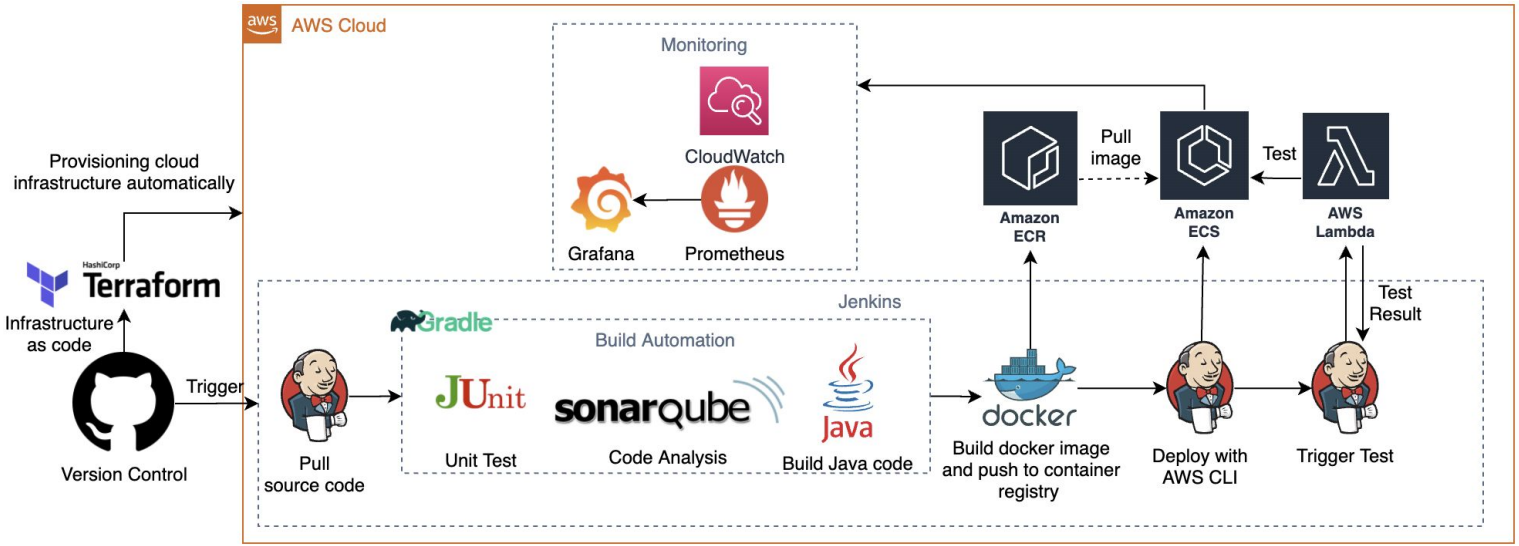
Research Approach

1. Investigate Serverless Offering in AWS

2. Develop Cloud-Based DevOps toolchain
   a. Non-integrated toolchain + Serverless computing services -> Experiment I & RQ1
   b. Integrated toolchain -> Experiment II

3. Experiments
   a. Experiment I : On non-integrated toolchain -> RQ1
   b. Experiment II :Compare integrated/non-integrated toolchain -> RQ2

Non-integrated toolchain - Overview



Problems in implementation:

- Lot of time and effort for:
  - Setting up cloud infrastructure
  - Configuration of Jenkins
- Lack of documentation, unstable Jenkins plugins
- Previous cloud knowledge needed

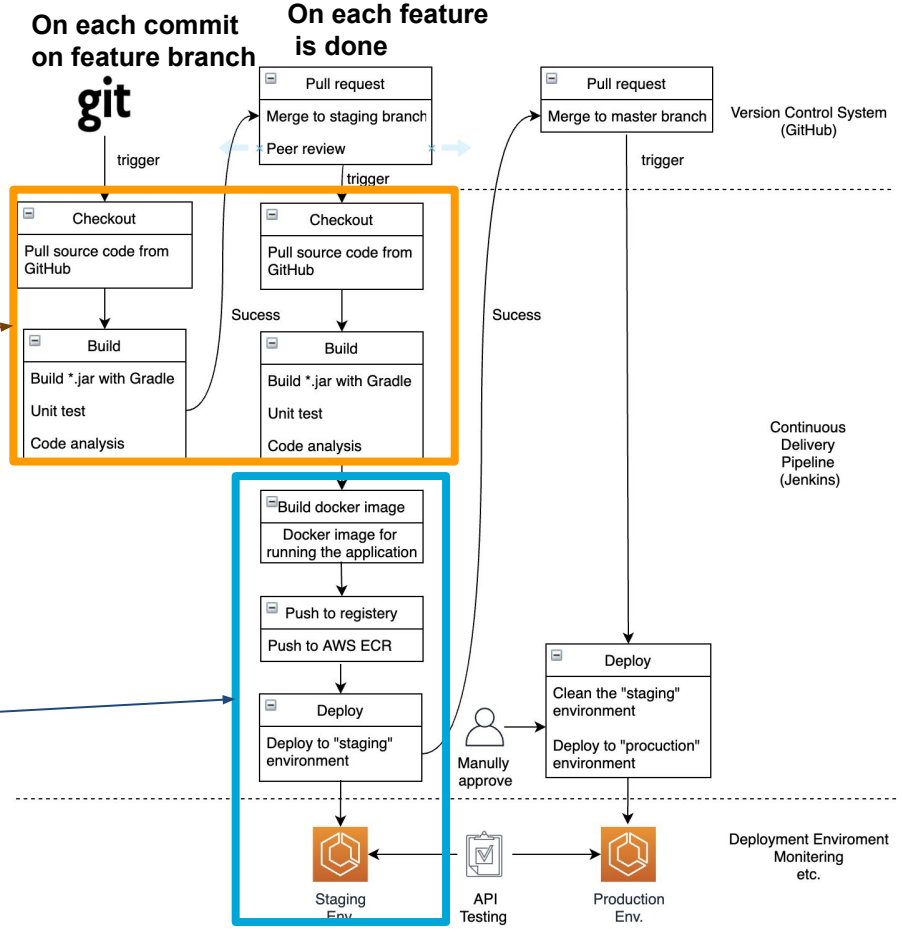
Non-integrated toolchain - Workflow

- Heavy, 80% pipeline runtime.
- More frequent.

Q: How to accelerate?

Why not run on feature branch?

- According to DevOps practice, developer should merge code often (several time a day)
- Already pretty frequent
- No need to run per commit

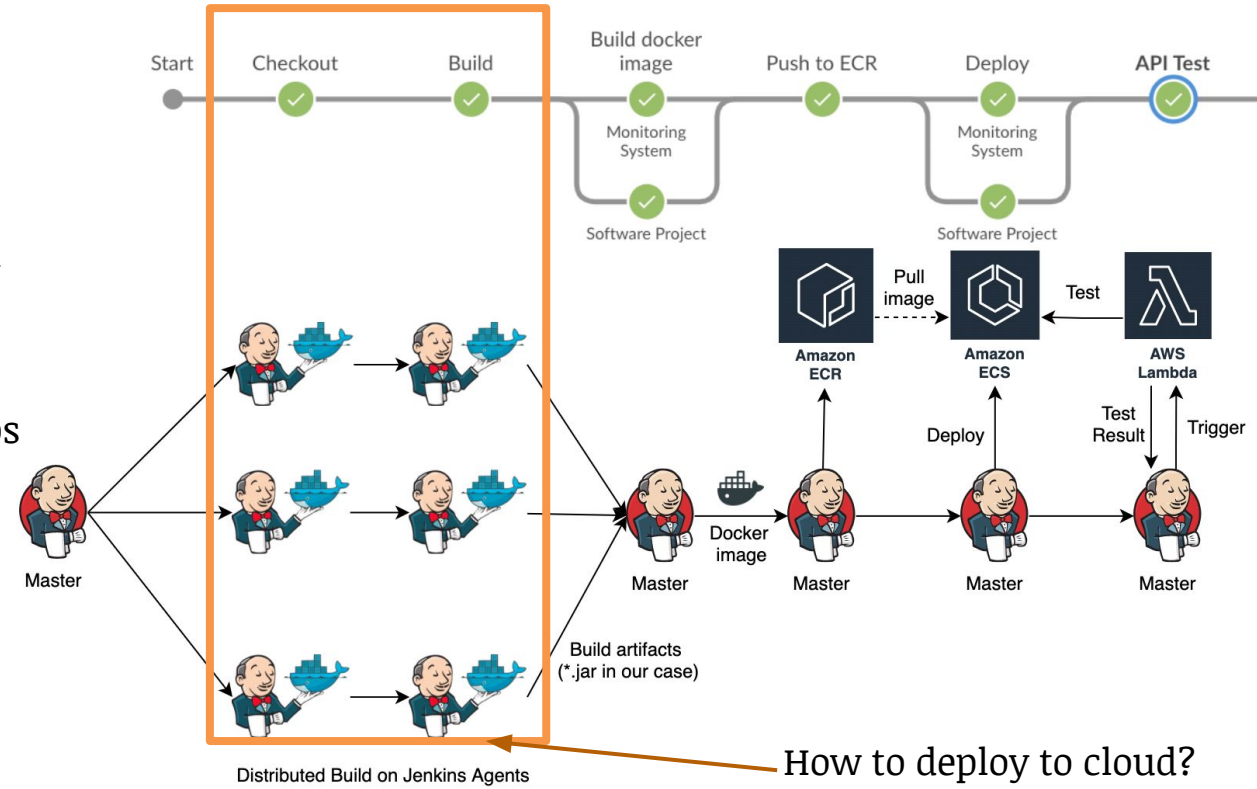


Non-integrated toolchain- Distributed Build

Solution:

Distributed Build with **build agents** run in **Docker containers**

- Parallel the heavy steps
- Why Docker? - Faster start, Easy to set up, Easy to change the build environment

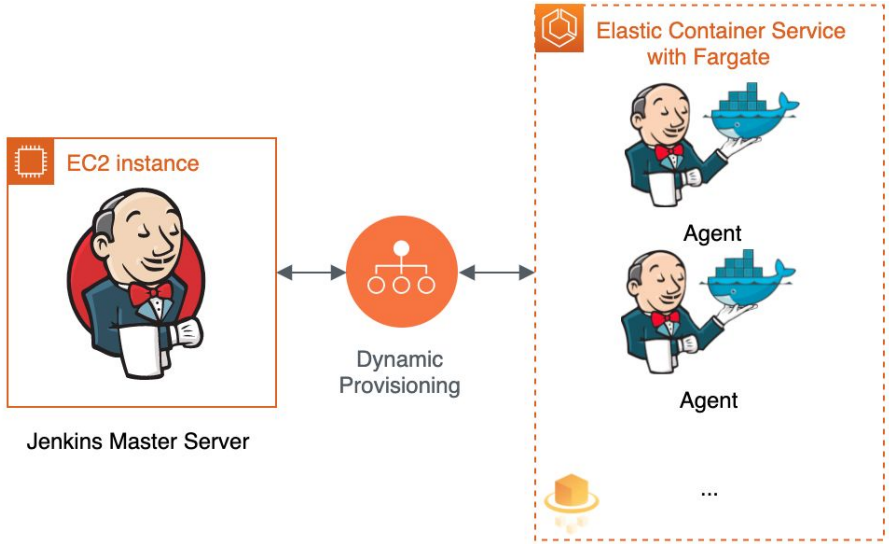


Using AWS serverless services - Fargate

Solution:

Hosting docker-based distributed build agent in **AWS Fargate**

- Master: Runs 24/7
  -> Traditional VM
- Jenkins agents:
  - Runs in Docker container
  - on demand, runs only when new job comes
    -> Serverless container services



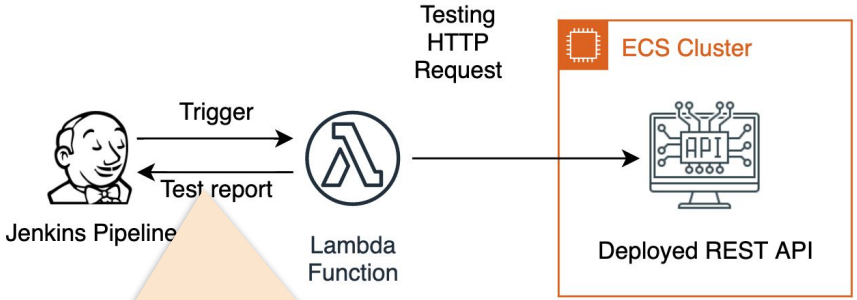
Using AWS serverless services - Lambda

Test deployed API with AWS Lambda

Manually?
- We need automation

VM?
- Complicated
- Pay for idle time

Lambda
- Event-driving
- Easy to set up
- Pay per use

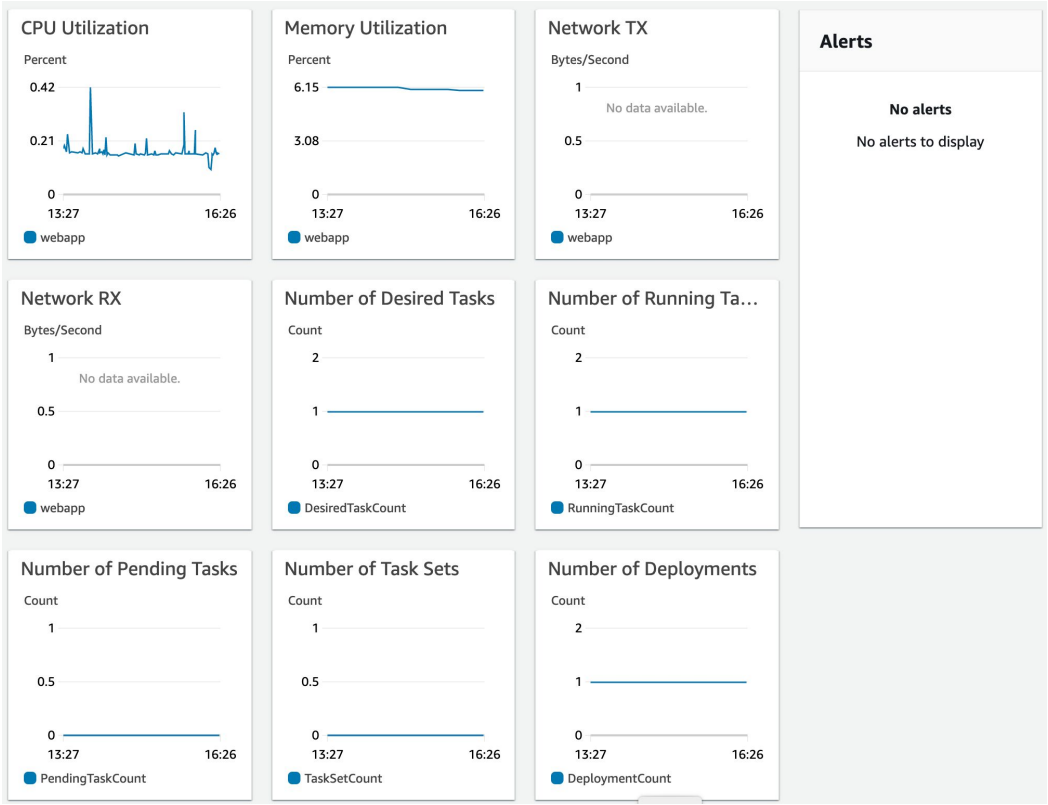



```
Testing the url http://bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com/
Testing the url http://bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com/packages
Get result of url: http://bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com/
1 out of 2 endpoints tested, 1 succeed
Get result of url: http://bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com/packages
2 out of 2 endpoints tested, 2 succeed
All endpoints are tested, result:
 {
  "http://bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com/": "Succeed, response body (first 300
characters): Hello world",
  "http://bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com/packages": "Succeed, response body (first
300 characters): [{\"name\":\"libws-commons-util-java\",\"description\":\"Common utilities from the
Apache Web Services Project\",\"dependencies\":[\"\"],\"reverseDependencies\":[]},{\"name\":\"python-pkg-
resources\",\"description\":\"Package Discovery and Resource Access using
pkg_resources\",\"dependencies\":[\"python (\u003e= 2.6)\",\"python (\u003c"
}
```

TU Delft

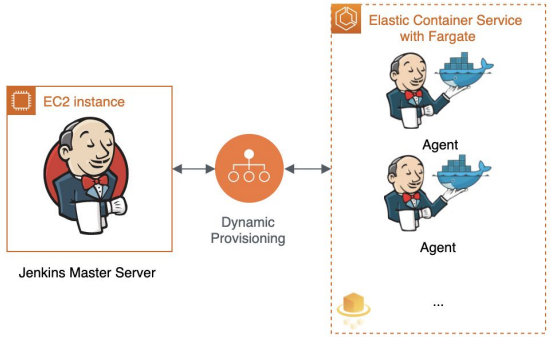
Using AWS serverless services - CloudWatch

Monitoring ECS cluster with AWS CloudWatch

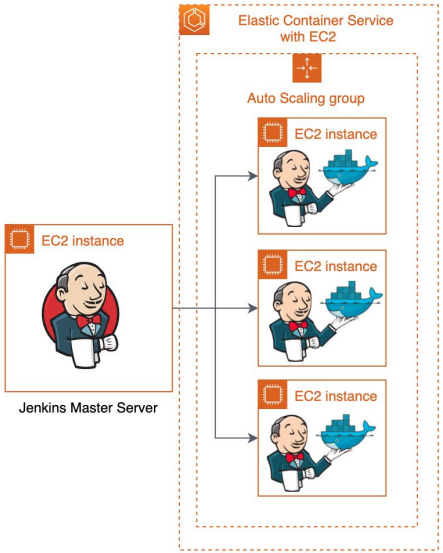


Experiment I - Fargate vs EC2

Be built at



VS



Case project:
A simple REST API web service developed by me with Spring Boot

Jenkins agents hosted by AWS Fargate

Jenkins agents hosted by a group of Auto-Scaling AWS EC2 virtual machines

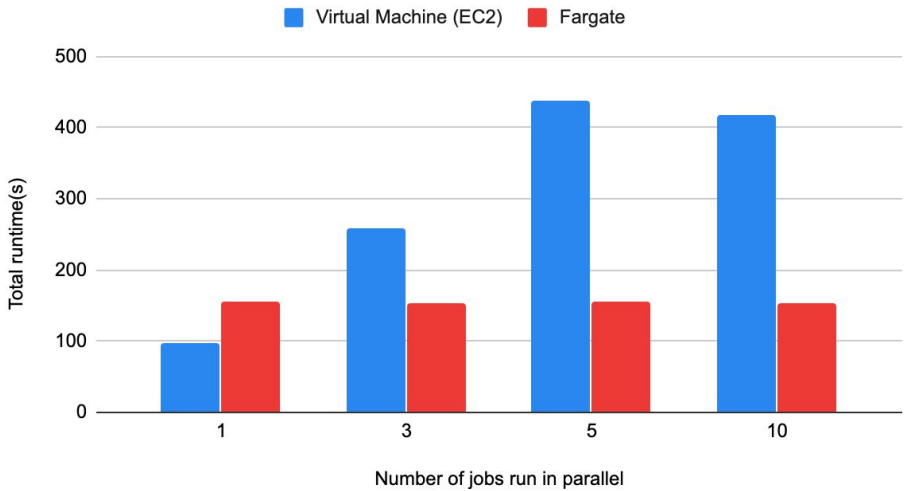
TU Delft

Experiment I - Result, Runtime

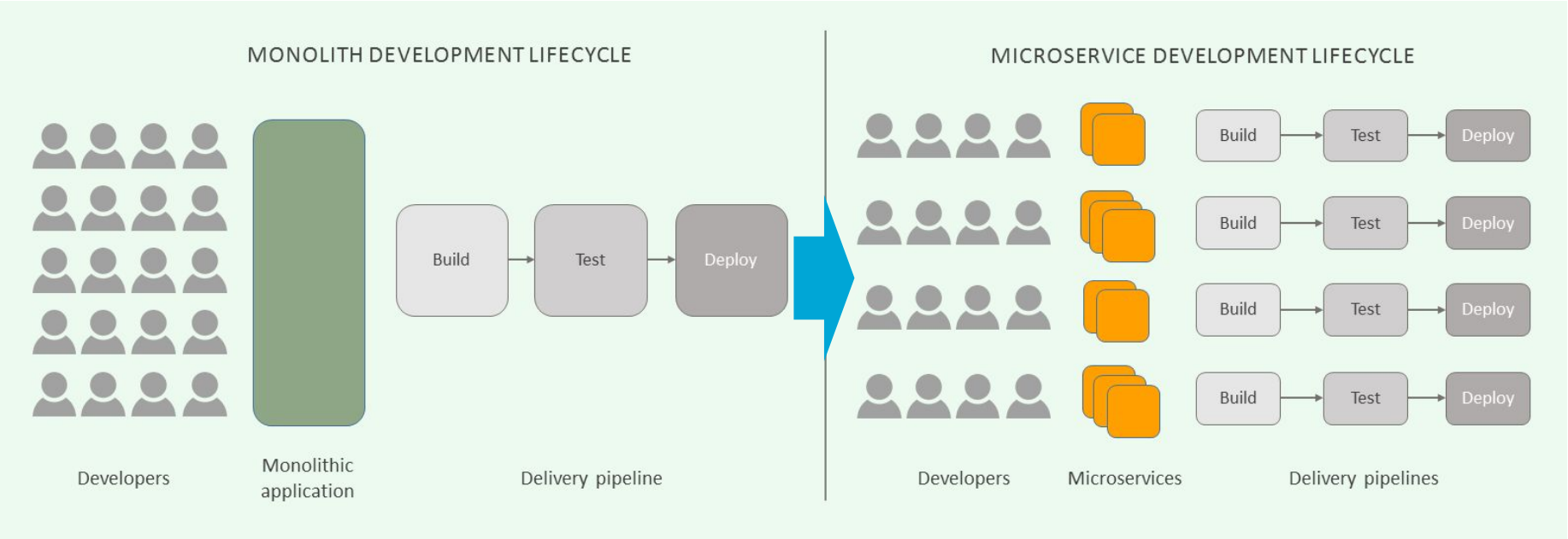
Performance with severless Jenkins agent:

2.8 X

under 5 tasks in parallel



Why Parallel performance is important?



Imaging having 100+ services in this microservices
DevOps Engineer:
🤯 Manage 100+ CI servers (Jenkins server in our case)
🤠 Manage one server, 100 + task runs in parallel

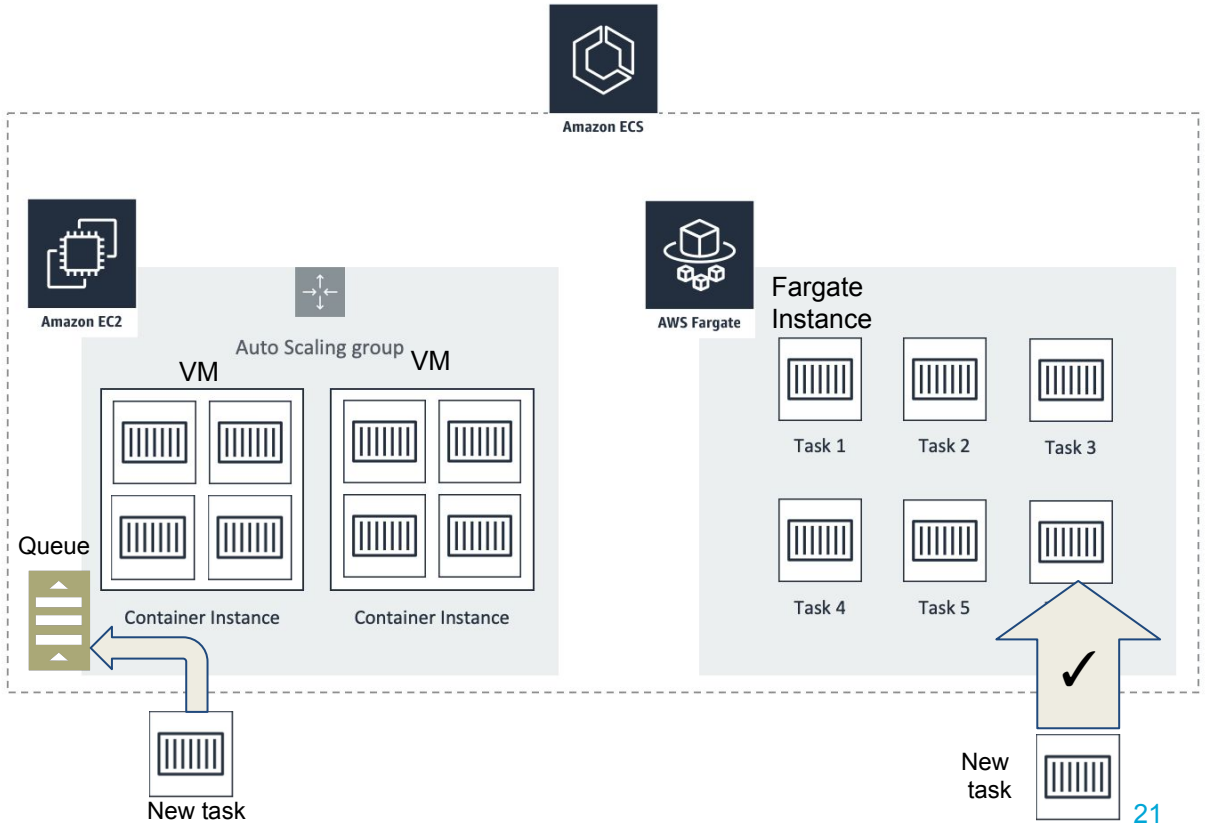
Experiment I - Result Evaluation, Runtime

EC2:

- Scaling is not related to the number of incoming tasks
- Competition for resource between agents on same VM
- Longer time to create new VM

Fargate:

- Unconditionally create a new instance
- Independent and isolated, no competition.
- Faster provision



Experiment I – Result, Cost & Resource Utilization

- Similar CPU/RAM utilization within container
- Fargate has higher cost per running hour
- No cost during Idle time - actual cost might lower

$0.0216 vs $0.0931

per hour runtime per agent, 2v CPU, 2 GB RAM

TU Delft

Integrated toolchain - Overview

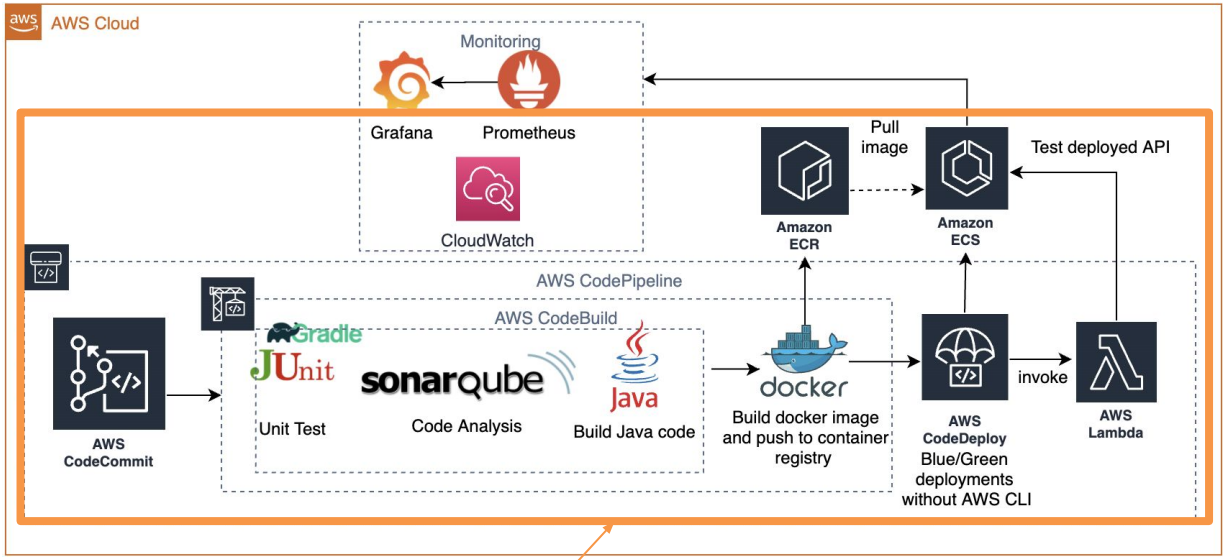
Compare with non-integrated toolchain:

Advantages:

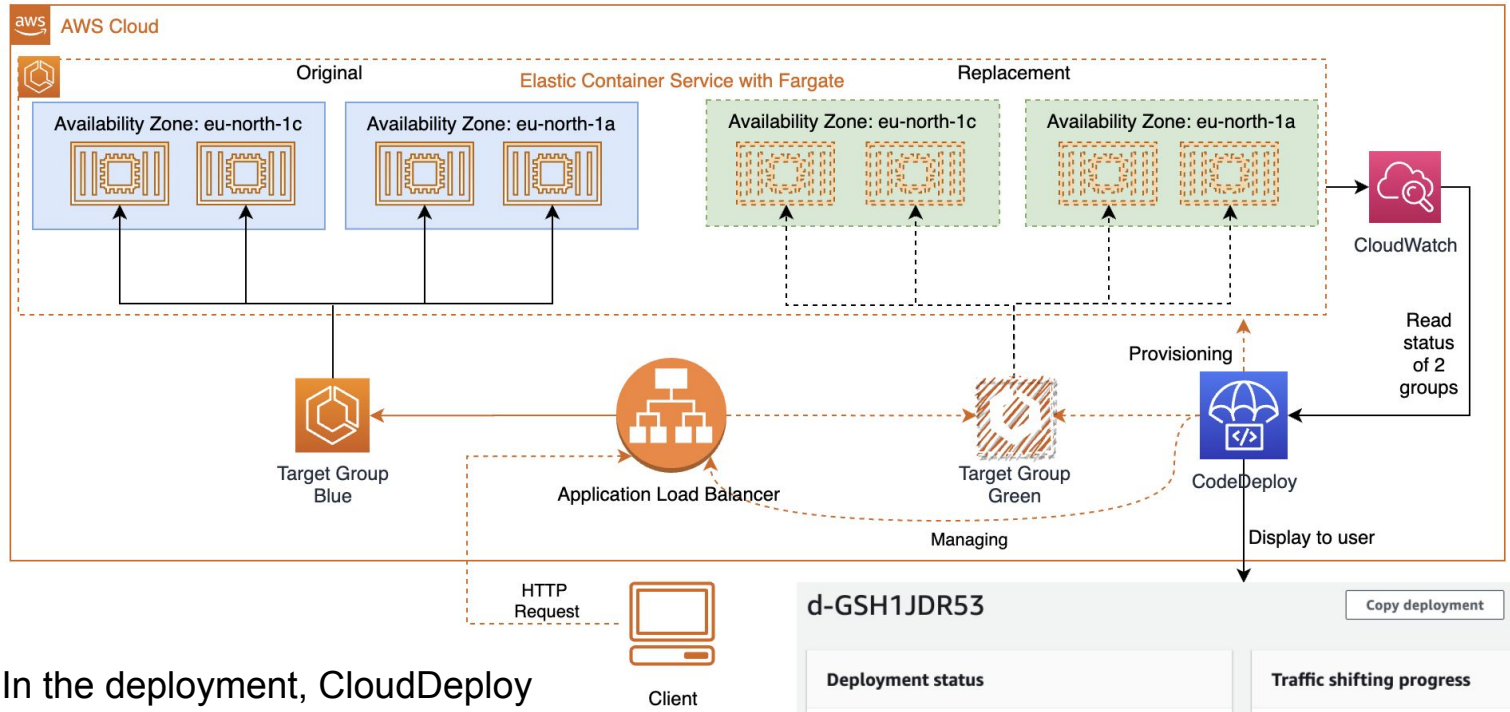
+ Less time and effort to set up
+ Better customer support
+ Better integration with AWS services -> **Blue/Green Deployment with CodeDeploy**

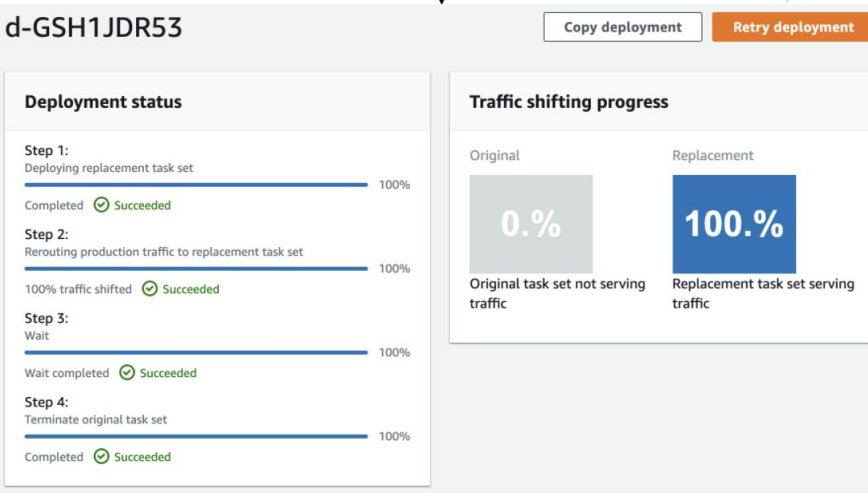
Disadvantages:

- Low visibility to infrastructure -> **Cannot identify error when CodeDeploy failed**
- Low freedom and extendibility -> **Third-party tools limited**



A single platform & serverless





In the deployment, CloudDeploy controls

- ALB
- Deployment Env. (ECS)
- Cloudwatch

TU Delft

Experiment II - Result, Runtime/Cost

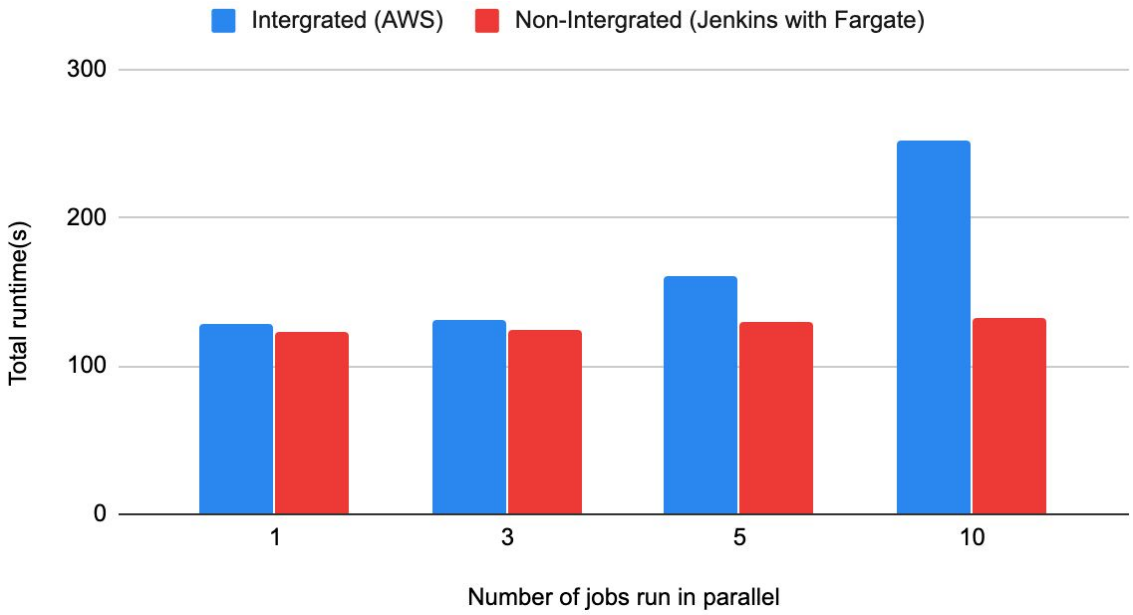
Evaluation on integrated/non-integrated DevOps toolchain

Non-integrated

- Better performance in parallel execution
- Lower cost, $0.1452/agent/hour

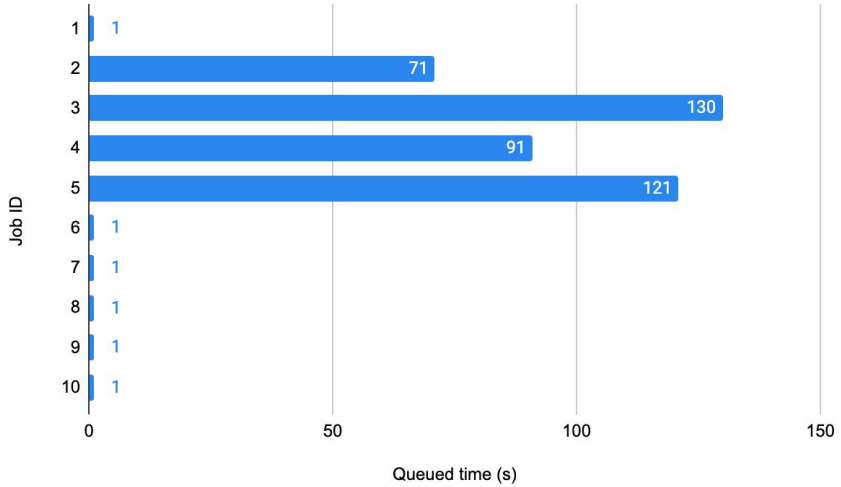
Integrated

- Lower performance in parallel execution ->Why (next slides)
- Higher per hour cost, $0.3/agent/hour. However, no need to pay the master node



Experiment II - Result analysis, Runtime

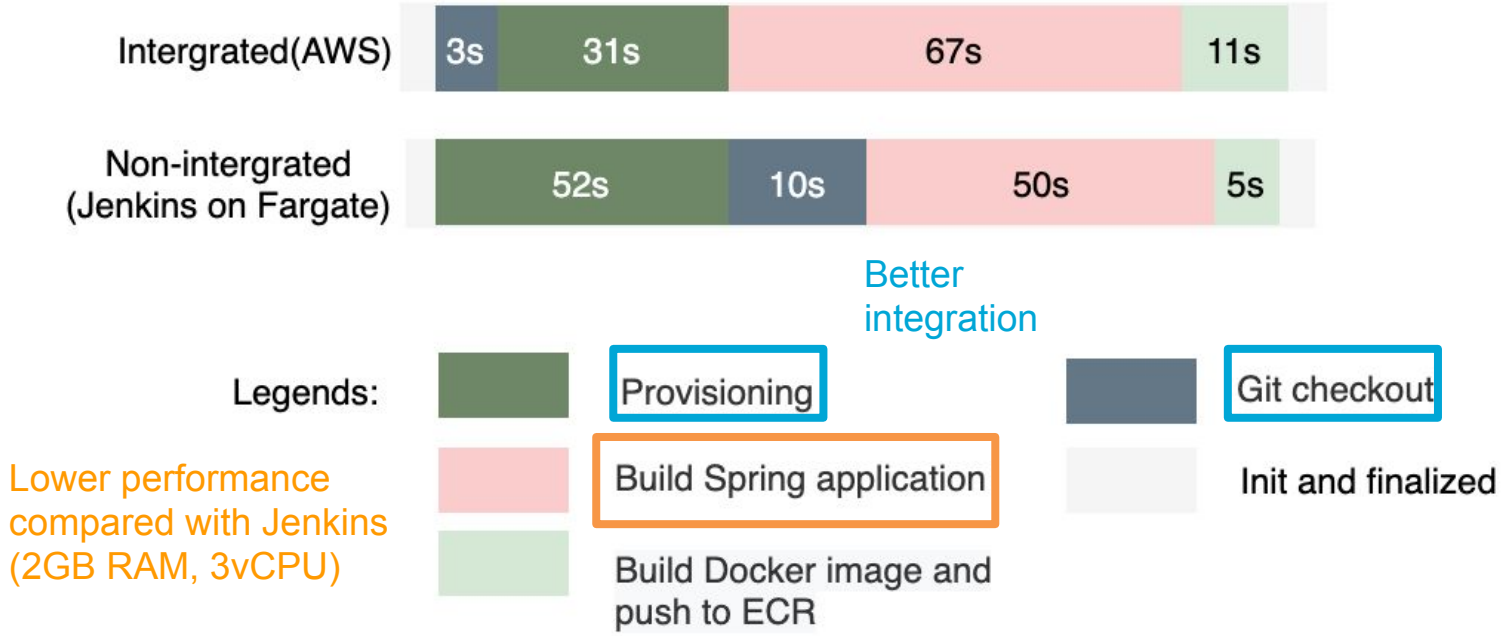
- AWS limiting resource during parallel build
- 4 out of 10 jobs have to queue



Build logs | Phase details | Reports | Environment variables | Build details

Name	Status	Context	Duration
SUBMITTED	Succeeded	-	<1 sec
QUEUED	Succeeded	-	61 secs
PROVISIONING	Succeeded	-	17 secs

Experiment II - Result analysis, Runtime



Decomposition of runtime of each stages in the pipeline

Conclusions

RQ1: How can serverless computing services in Amazon Web Services enhance the DevOps toolchain?

- Three possible services
  - Fargate for hosting distributed agents
    - Better parallel execution performance good for microservice development and larger team.
    - Higher hourly cost, total cost might not higher
  - CloudWatch for monitoring solution
  - Lambda for event driving task (e.t. integration test of deployed API)
    - Less maintenance effort
    - Lower cost

- In conclusion:
  Improving performance, eliminating server management tasks, and reducing the cost of idle time.

TU Delft

Conclusions

RQ2: How does the integrated toolchain build with AWS DevOps Tools compare with the traditional non-integrated toolchain?

- Performance: Lower than Jenkins with agent in Fargate
  - Resource limitation -> long queueing time
  - Longer build time under same hardware.
- Cost: Higher per hour price, however no cost for idle time
- Development: Easy integration with other AWS services, less time and effort to develop and better customer support. Low visibility to the underlying infrastructure.

Future Works

- **More evaluation method on toolchains**
  i.e. Have two software team in real-life test the toolchains.

- **More complicated case project in experiments**
  - Could be a microservice with different type of services
    i.e. Database, REST API, Lambda functions, etc.
  - Strength our conclusion

Thanks for your attention!

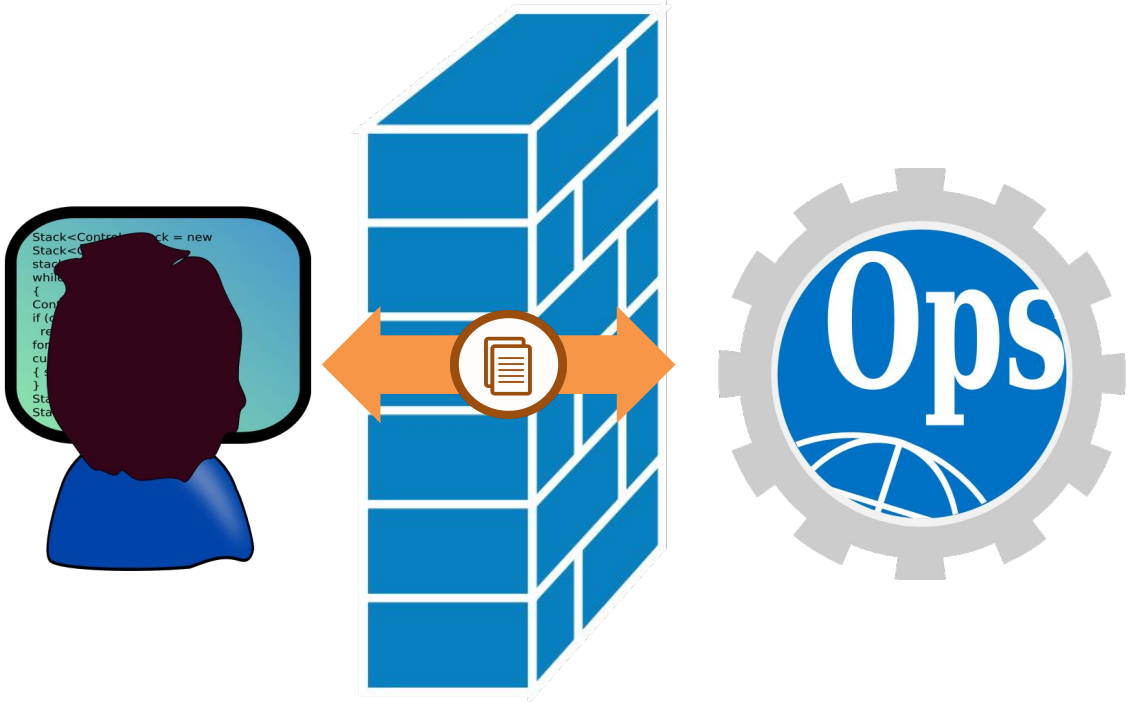
Q & A

TU Delft

The following pages are skipped

3mins, problem / RQs
10 mins p1
8 mins p2
4 mins conclusion

What is DevOps?



AWS Serverless platform

- AWS Fargate: Run container serverless
- AWS Lambda: Run code serverless, could be triggered by AWS events
- AWS CloudWatch: Monitoring application
- AWS DevOps Tools:
  - CodeCommit: Git version control; cooperation.
  - CodeBuild: Build Automation
  - CodeDeploy: Deploy application to AWS/ on-premise servers, support advanced deploy methodology
  - CodeStar: Connecting above services

Agenda

01
Background

02
Problem
Statement

03
Implementation

04
Experiments

05
Result

06
Conclusion

TU Delft

About Me

Aalto-yliopisto
Aalto-universitetet
Aalto University

TU Delft
Delft University of Technology

eit Digital MASTER SCHOOL

EIT Digital Double Degree
Cloud Computing and Services

DevOps Consultant
Eficode
Helsinki, Finland 🇫🇮

TU Delft

What is DevOps?

A set of practices aims to combine traditionally separated disciplines (DEV + OPS), intended to speed up software delivery, while ensuring high quality.

DevOps - DevOps Toolchain and DevOps Practises

Infrastructure as Code

Project Management & Planning

Version Control

Build Automation

Continuous Integration & Delivery

Monitoring

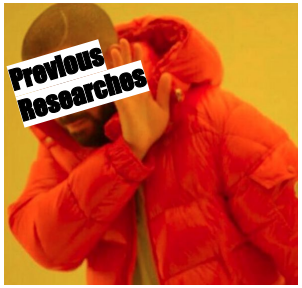
Test Automation

TU Delft

Problem: There is not enough research on how serverless can be used in DevOps toolchain

DevOps for Serverless Application?

1. DevOps for Serverless Applications: Design, deploy, and monitor your serverless applications using DevOps practices (Shashikant Bangera, 2018)
2. Implementation of a DevOps pipeline for serverless applications (Ivanov, Vitalii and Smolander, Kari, 2018)



Use Serverless in DevOps?

1. Several blogs such as “The Serverless Path to DevOps” (Sharjeel Yusuf, Feb 27, 2020)
2. No directly related paper so far

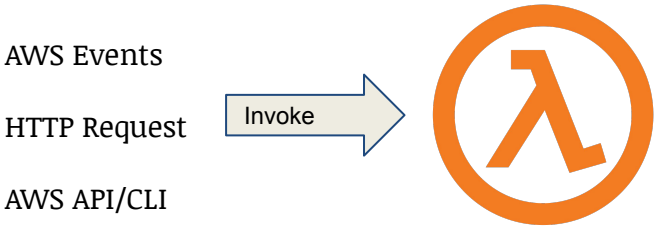
Services in AWS - Elastic Container Service(ECS) with AWS Fargate

ECS: Container orchaction services.

Could runs Docker containers in:

- EC2 cluster
- Fargate -- Serverless service for Docker containers

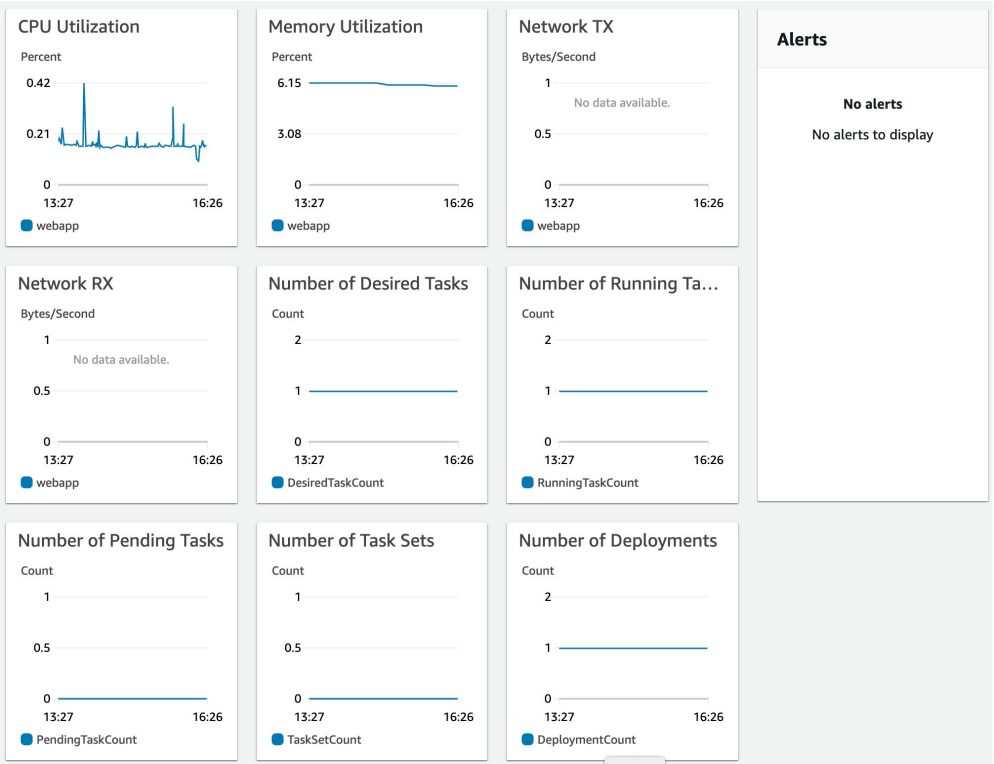
Services in AWS – AWS Lambda

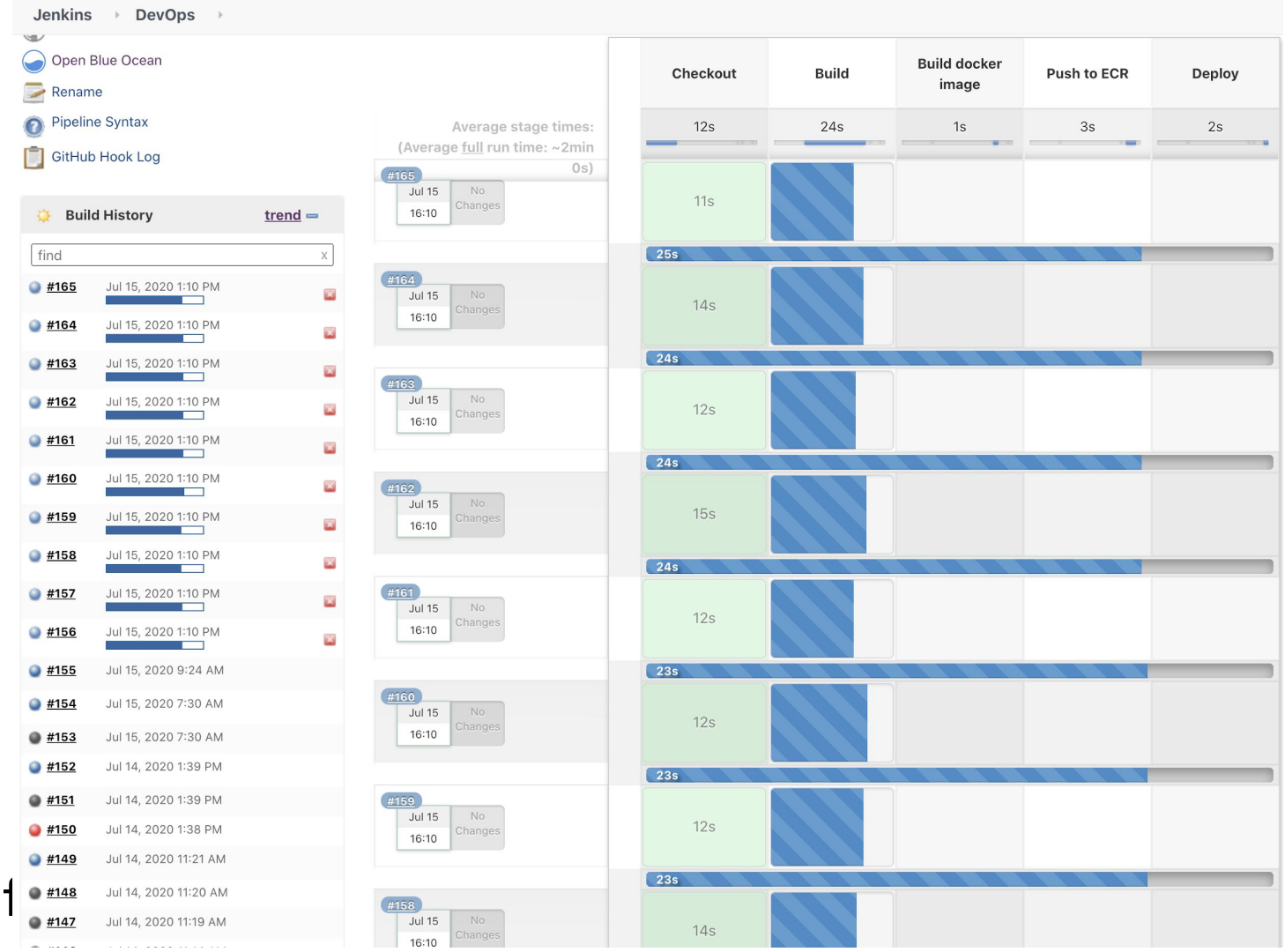


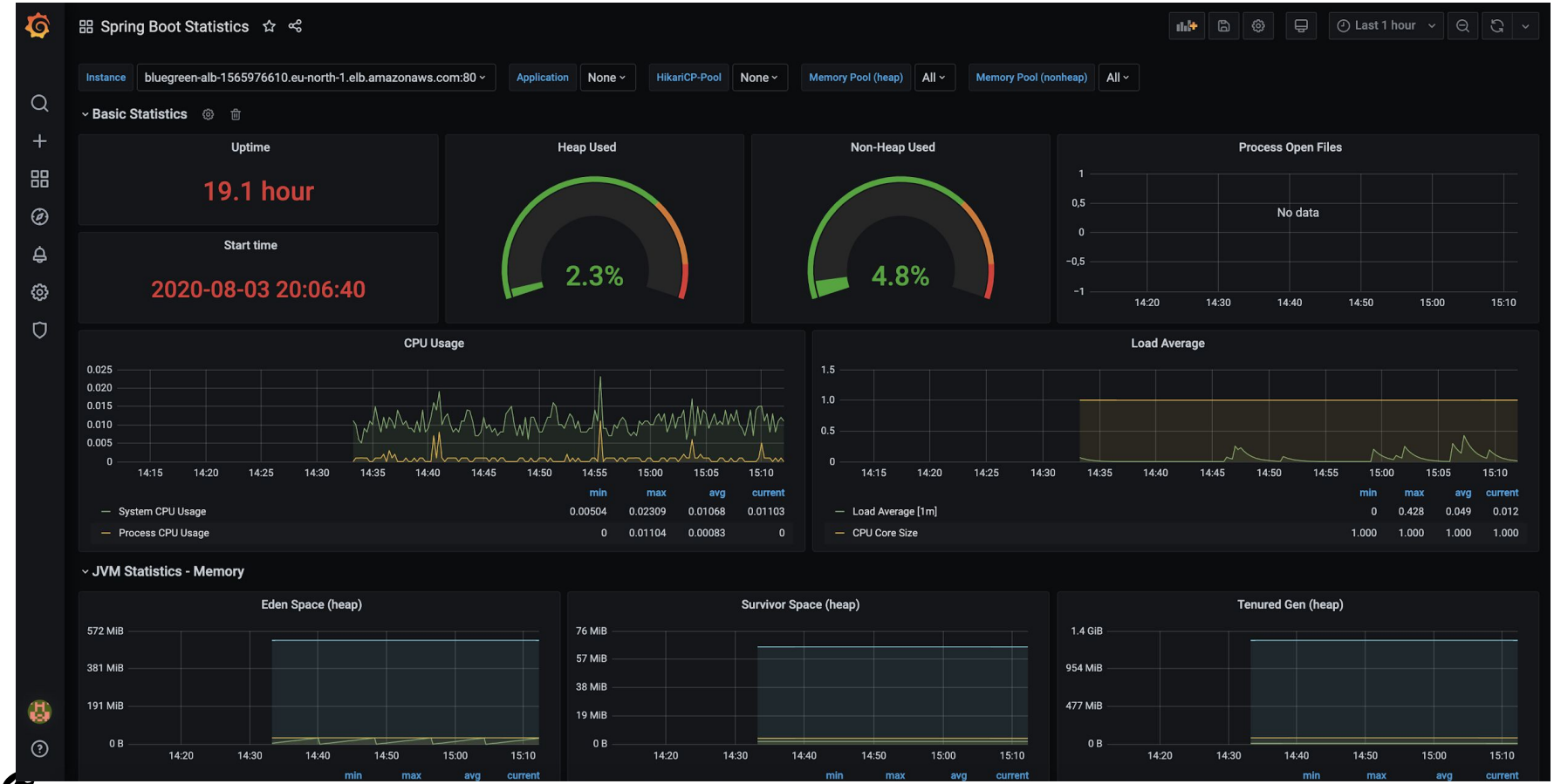
- Run code without thinking about servers
- Event Driving
- Pay-as-you-go

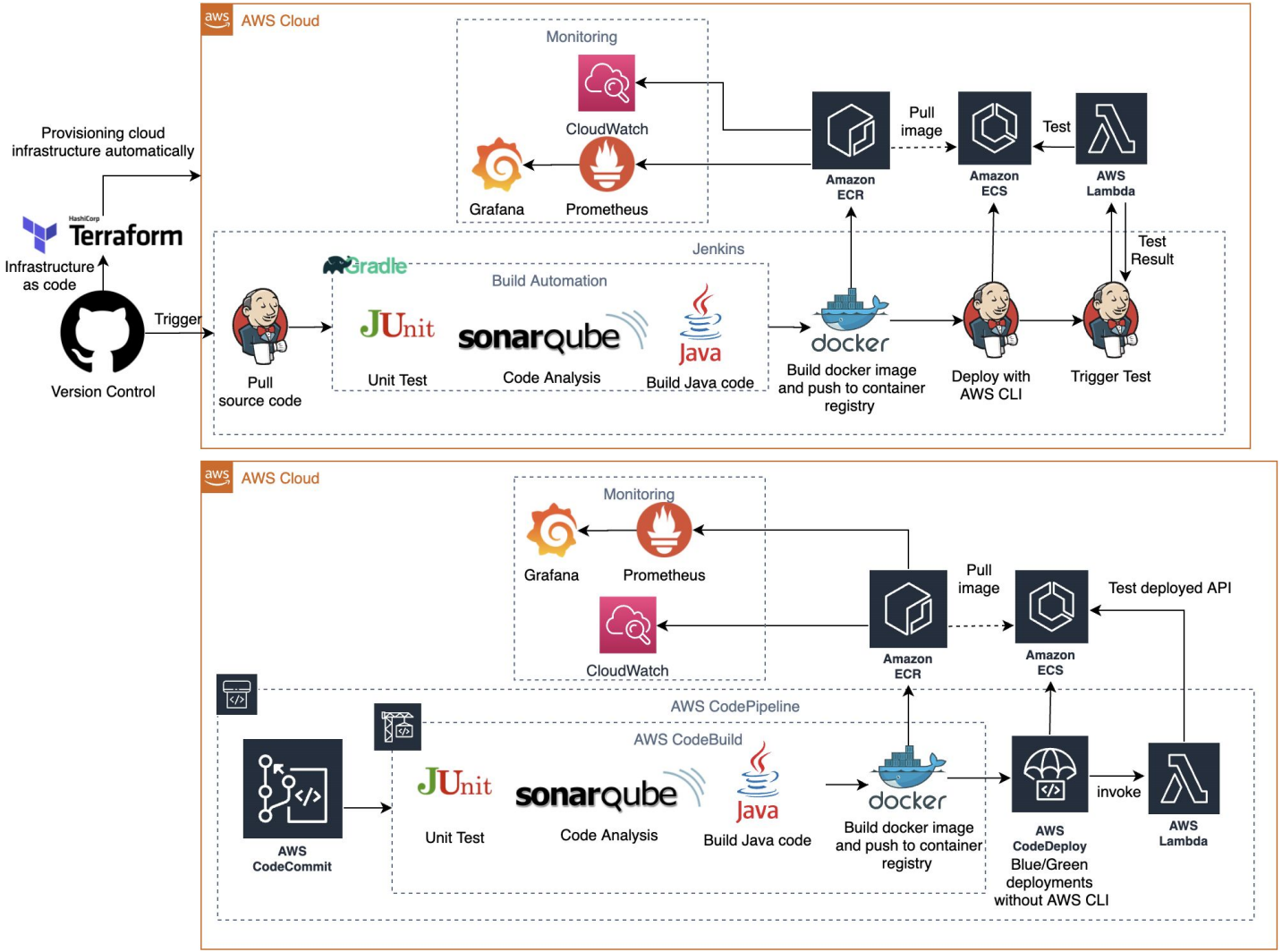
Services in AWS – AWS CloudWatch

- Out-of-the-box
- Monitoring deployed application in AWS
- Logs and metrics



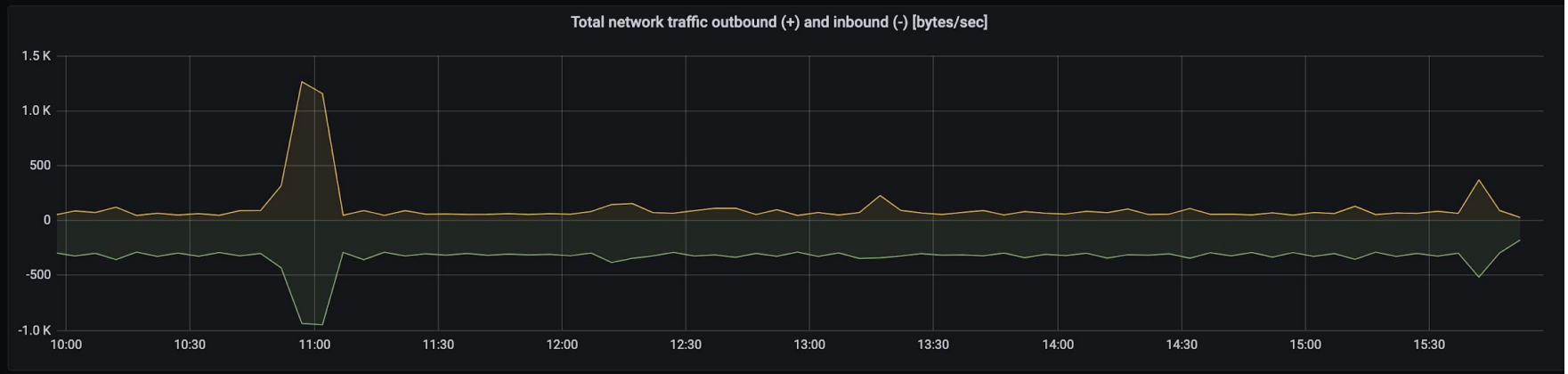

Jenkins
DevOps
Open Blue Ocean
Rename
Pipeline Syntax
GitHub Hook Log
Build History
trend
find
#165 Jul 15, 2020 1:10 PM
#164 Jul 15, 2020 1:10 PM
#163 Jul 15, 2020 1:10 PM
#162 Jul 15, 2020 1:10 PM
#161 Jul 15, 2020 1:10 PM
#160 Jul 15, 2020 1:10 PM
#159 Jul 15, 2020 1:10 PM
#158 Jul 15, 2020 1:10 PM
#157 Jul 15, 2020 1:10 PM
#156 Jul 15, 2020 1:10 PM
#155 Jul 15, 2020 9:24 AM
#154 Jul 15, 2020 7:30 AM
#153 Jul 15, 2020 7:30 AM
#152 Jul 14, 2020 1:39 PM
#151 Jul 14, 2020 1:39 PM
#150 Jul 14, 2020 1:38 PM
#149 Jul 14, 2020 11:21 AM
#148 Jul 14, 2020 11:20 AM
#147 Jul 14, 2020 11:19 AM
Average stage times:
(Average full run time: ~2min 0s)
Checkout
Build
Build docker image
Push to ECR
Deploy
12s
24s
1s
3s
2s
#165 Jul 15 16:10 No Changes
11s
25s
#164 Jul 15 16:10 No Changes
14s
24s
#163 Jul 15 16:10 No Changes
12s
24s
#162 Jul 15 16:10 No Changes
15s
24s
#161 Jul 15 16:10 No Changes
12s
23s
#160 Jul 15 16:10 No Changes
12s
23s
#159 Jul 15 16:10 No Changes
12s
23s
#158 Jul 15 16:10 No Changes
14s


Spring Boot Statistics
Last 1 hour
Instance
bluegreen-alb-1565976610.eu-north-1.elb.amazonaws.com:80
Application
None
HikariCP-Pool
None
Memory Pool (heap)
All
Memory Pool (nonheap)
All
Basic Statistics
Uptime
19.1 hour
Start time
2020-08-03 20:06:40
Heap Used
2.3%
Non-Heap Used
4.8%
Process Open Files
No data
CPU Usage
min max avg current
System CPU Usage 0.00504 0.02309 0.01068 0.01103
Process CPU Usage 0 0.01104 0.00083 0
Load Average
min max avg current
Load Average [1m] 0 0.428 0.049 0.012
CPU Core Size 1.000 1.000 1.000 1.000
JVM Statistics - Memory
Eden Space (heap)
Survivor Space (heap)
Tenured Gen (heap)


AWS Cloud
Monitoring
CloudWatch
Grafana
Prometheus
Amazon ECR
Pull image
Amazon ECS
Test
AWS Lambda
Provisioning cloud infrastructure automatically
Terraform
Infrastructure as code
Version Control
Trigger
Jenkins
Gradle
Build Automation
JUnit
Unit Test
sonarqube
Code Analysis
Java
Build Java code
Pull source code
docker
Build docker image and push to container registry
Deploy with AWS CLI
Trigger Test
Test Result
AWS Cloud
Monitoring
Grafana
Prometheus
CloudWatch
Amazon ECR
Pull image
Amazon ECS
Test deployed API
AWS CodePipeline
AWS CodeBuild
AWS CodeCommit
JUnit
Unit Test
sonarqube
Code Analysis
Java
Build Java code
docker
Build docker image and push to container registry
AWS CodeDeploy
Blue/Green deployments without AWS CLI
invoke
AWS Lambda

Amazon EC2

Last 6 hours



Network details