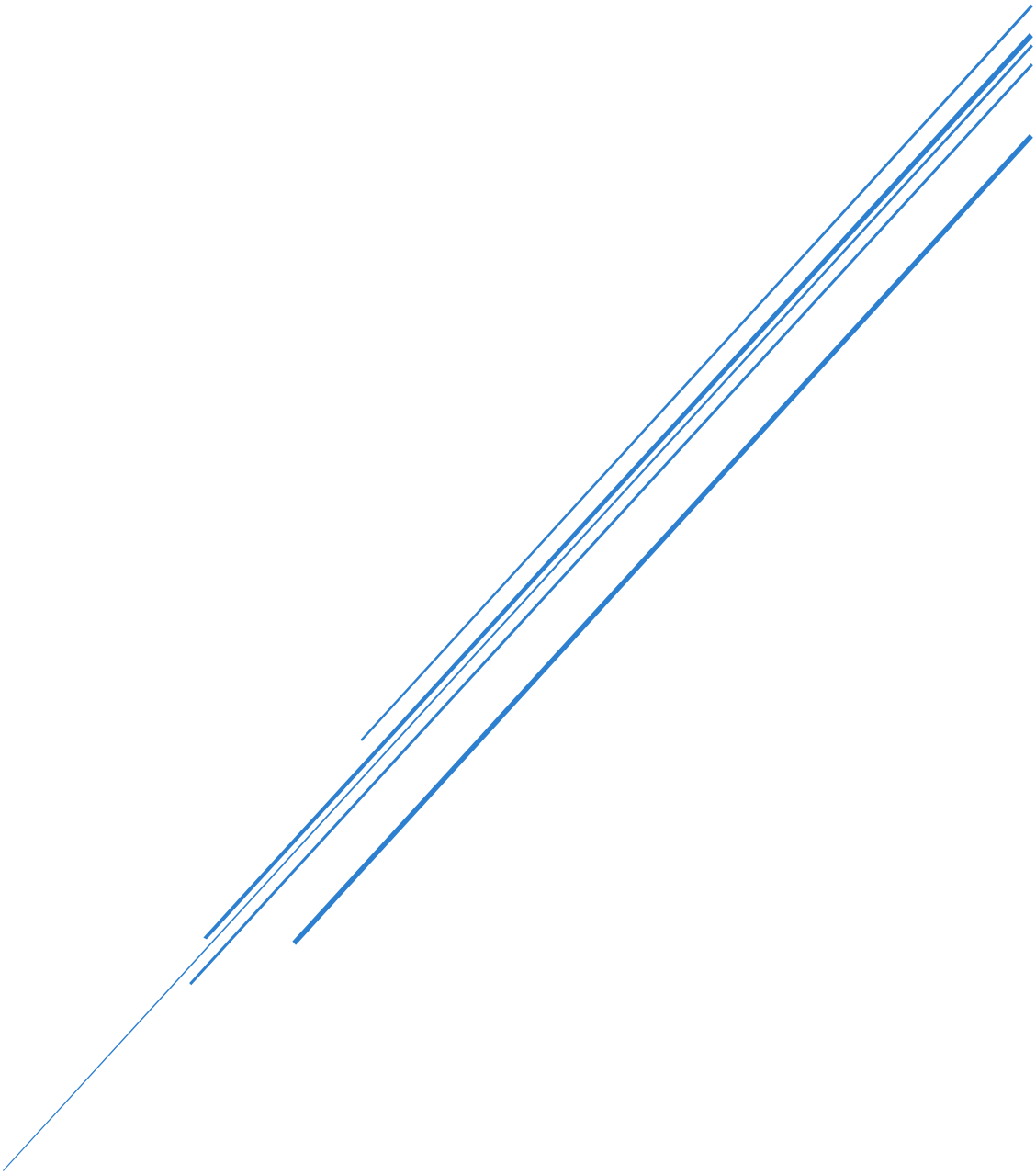
DEVOPS DOCUMENTATIE



Jari Knoop, Dorian Baies en Kim Nguyen

21-6-2024

INHOUDSOPGAVE

PLAN DEVOPS

Groep: Jari Knoop, Dorian Baies, Kim Nguyen

DEVOPS IN PROJECT

Voor het vak DevOps willen wij zo gestroomlijnd en efficiënt mogelijk te werken gaan. Met software dat goed samenwerkt. En voor een goede operationele cyclus te creëren. Daarom hebben wij de volgende keuzes gemaakt:

1. **Plan**: Voor het vak DevOps en ons project willen wij Azure Devops gebruiken voor het beheren van backlogs en sprints. Hierdoor kunnen wij overzichtelijk onze user-stories en doelen opstellen.

2. **Code**: Voor het bewaren en updaten, toevoegen en verwijderen van code gaan wij GitHub gebruiken. GitHub integraties bieden ons flexibiliteit en gemak bij het opzetten van onze workflow en repositories. Ook kan GitHub goed samenwerken met Docker voor onze build van het project.

3. **Build**: Docker wordt gebruikt om images en containers aan te maken. Docker containers isoleren de applicatie en alle benodigde afhankelijkheden van de hostomgeving. Dit voorkomt conflicten tussen verschillen projecten en maakt het mogelijk om de applicatie op meerdere platforms te runnen.

4. **Test**: Azure Testplans is een tool binnen Azure DevOps en biedt een gebruiksvriendelijke manier voor het testen van de applicatie. Het maakt het mogelijk om geautomatiseerde testen uit te voeren en handmatige testen te plannen en uit te voeren, wat helpt aan de kwaliteit van de applicatie en fouten kan voorkomen bij deployment.

5. **Deploy**: Azure Pipelines automatiseert de implementatie van de applicatie. Hierdoor is het proces van bouwen en productie overzichtelijk.

6. **Operate**: Kubernetes wordt gebruikt in de operationele omgeving, voor het schalen en beheren van containers. Hierdoor kan de container makkelijk worden geschaald.

7. **Monitor**: Grafana is om de omgeving inzichtelijk te krijgen, zodat we inzicht krijgen in hoe de applicatie functioneert. Dit helpt bij het opsporen van problemen.

We gaan Azure DevOps gebruiken voor de meeste stappen, zodat alles bij elkaar staat. Docker is voor het creëren van images en containers. Grafana is voor een veilige en stabiele omgeving om operationele informatie in beeld te krijgen.

PROJECT KEUZE

We kiezen het project van SL2B (Software Language 2b), omdat het een nieuw en lopend project is wat naast DevOps gegeven wordt in de 4de periode, daarnaast kunnen er in het project van SL2B vanaf het begin DevOps concepten worden toegepast. Dit biedt ons de perfecte mogelijkheid om onze kennis betreft DevOps in te zetten voor een juiste deployment en alle andere componenten.

Tijdens de productie van SL2B kan DevOps onderdelen zoals het opstellen van een plan via Azure DevOps om een planning bij te houden een goede keuze zijn. Ook om bijvoorbeeld de code te delen via GitHub om regelmatig nieuwe functionaliteiten toe te voegen of om code te bewerken. Ook zal het helpen met de structuur van het project. Met SL2B leren we niet alleen nieuwe codeerconcepten, maar kunnen we die concepten ook goed verwerken met Docker om zo het project klaar te maken voor meerdere platforms en stakeholders.

Het project SL2B bevat de perfecte onderdelen die verwerkt kunnen worden met DevOps om de deployment zo goed te mogelijk te laten verlopen met behulp van alle praktijken en technieken die in het eerste hoofdstuk staan benoemd. Omdat SL2B en DevOps zo goed aansluiten biedt het ons junior softwareontwikkelaars een mooie mogelijkheid om de nieuwe kennis van DevOps toe te passen binnen het project.

Tot slot willen wij Scrum toepassen binnen het project. Met behulp van Scrum kunnen wij makkelijker het project managen en plannen. Zo gaan wij binnen het team sprints planningen en user-stories opstellen voor de eisen van het project. Dit zorgt ervoor dat er meer overzicht komt en iedereen weet wat er van hem of haar verwacht wordt. Deze sprint en taken zullen regelmatig terugkoppeling krijgen van de stakeholders en feedback opgegeven binnen het team tijdens de sprint reviews. Ook helpt de Scrum techniek met het verdelen van de taken en iedereens sterke en zwakke punten te benutten voor het project. Tijdens de ontwikkeling van het project kunnen we van diverse vaardigheden gebruik maken, zoals: ontwikkelen, testen en documenteren.

AANDACHTSPUNTEN EN VERBETERPUNTEN

We hebben eerder groepsprojecten moeten doen, hieruit kunnen we verbeterpunten opnoemen om dit project beter uit te voeren. Hieronder wordt beschreven hoe we dit concreet gaan aanpakken.

Aandachtpunt: Een duidelijk takenverdeling met beschrijving voor ieder teamlid
Als de tussentijdse opdrachten bekend zijn na het college, dan gaan we de taken verdelen in onze groep. Er zal een beschrijving zijn van wat de taken inhouden in onze planning, wat te zien zal zijn in onze Discord server. We streven ernaar de tussentijdse opdrachten af te hebben voor de volgende les. Wanneer een groepslid klaar is met een taak, dan uploadt degene zijn werk naar GitHub, zodat anderen erbij kunnen en verder kunnen werken als ze een bepaald onderdeel van iemand anders nodig hebben.

Aandachtpunt: Uitgewerkte backlogs verdelen over sprints en slim prioriteren
Voordat we gaan programmeren gaan we eerst de backlog uitwerken met user stories die functionele acceptatiecriteria en non-functionele criteria bevatten, hierdoor is het duidelijk wat het inhoudt en wanneer een user story af is. Na de goedkeuring van onze docenten, gaan we de backlog verdelen over de sprints, elke sprint duurt twee weken en ons doel is om 1 week voor de oplevering het project af te hebben. Om dit te bereiken gaan we beginnen met de functionaliteiten die de hoogste prioriteit hebben en kijken we daarna welke andere user stories of functionaliteiten er mee te maken hebben. Ook kijken we naar welke functionaliteiten noodzakelijk eerst moeten worden gedaan voordat er kan worden gewerkt aan de volgende functionaliteit. Met deze overwegingen gaan we de backlog die bij elkaar horen over de sprints verdelen. Aan het einde van de sprint kijken we terug op welke user stories af zijn en welke eventueel verbetering nodig hebben. Deze kennis nemen we mee naar de daaropvolgende sprint.

Met deze backlogverdeling kunnen we even veel aandacht besteden aan de backend en frontend, want alle belangrijke en noodzakelijke user stories worden eerst gedaan. Als deze af zijn en er tijd over, dan besteden we aandacht aan de user stories die een lagere prioriteit hebben.

UNIT TESTS

ADD ANIMAL

User Story Titel: Add Animal

Beschrijving: Deze unit test valideert dat een dier succesvol kan worden toegevoegd aan een verblijf in een zoo-managementsysteem. De test maakt een nieuw verblijf aan en voegt een dier met de naam "Lion" en een unieke ID toe aan dit verblijf. Vervolgens controleert de test of het dier daadwerkelijk in de lijst van dieren in het verblijf aanwezig is.

Acceptatiecriteria:

- Een nieuw verblijf (Enclosure) moet correct worden aangemaakt.
- Een nieuw dier (Animal) met een unieke ID en naam moet correct worden aangemaakt.
- Het dier moet succesvol kunnen worden toegevoegd aan de dierenlijst van het verblijf.
- Na toevoeging moet het dier in de dierenlijst van het verblijf aanwezig zijn.

Specificaties en Requirements:

Specifiaties:

1. Moet een klasse zijn die een collectie van dieren (Animals) bevat.
2. De Animals collectie moet bij initialisatie leeg zijn.
3. Een unieke identifier voor het dier.
4. De naam van het dier.
5. Het verblijf (Enclosure) moet de mogelijkheid bieden om dieren (Animal) toe te voegen aan de Animals collectie.
6. Na toevoeging moet het dier beschikbaar zijn in de Animals collectie van de Enclosure.

Requirements:

1. Wanneer een Enclosure object wordt aangemaakt, moet dit object een lege Animals collectie bevatten.
2. Een Animal object moet kunnen worden aangemaakt met een unieke Id en een Name.
3. De Enclosure klasse moet een methode of mechanisme bevatten waarmee een Animal object kan worden toegevoegd aan de Animals collectie.
4. Na toevoeging moet het mogelijk zijn om te verifiëren dat het Animal object daadwerkelijk in de Animals collectie van de Enclosure aanwezig is.

Test Cases: Dier toevoegen aan Enclosure

- Beschrijving: Controleer of een Animal object succesvol kan worden toegevoegd aan de Animals collectie van een Enclosure.
- Input: Voeg het Animal object toe aan de Animals collectie van een Enclosure object.
- Verwachte Uitkomst: Het Animal object moet aanwezig zijn in de Animals collectie van de Enclosure.

REMOVE ANIMAL

User Story Titel: Remove Animal

Beschrijving: Deze unit test controleert de functionaliteit van het verwijderen van een dier uit een verblijf. Een nieuw verblijf (Enclosure) wordt gecreëerd en een dier (Animal) met de naam "Lion" en een unieke ID wordt toegevoegd aan dit verblijf. Daarna wordt het dier weer verwijderd uit het verblijf. De test verifieert vervolgens of het dier niet meer in de lijst van dieren van het verblijf aanwezig is.

Acceptatiecriteria:

- Een nieuw Enclosure object moet correct worden aangemaakt en een lege Animals collectie bevatten.
- Een nieuw Animal object moet kunnen worden aangemaakt met een unieke Id en een niet-lege Name.
- Het dier moet succesvol aan de Animals collectie van het Enclosure object kunnen worden toegevoegd en daarna verwijderd.
- Na verwijdering moet het dier niet langer in de Animals collectie van het Enclosure object aanwezig zijn.

Specificaties en Requirements:

Specificaties:

1. Moet een klasse zijn die een collectie van dieren (Animals) bevat.
2. De Animals collectie moet bij initialisatie leeg zijn.
3. Een unieke id voor het dier.
4. De naam van het dier.
5. Het verblijf (Enclosure) moet de mogelijkheid bieden om dieren (Animal) te verwijderen uit de Animals collectie.
6. Na verwijdering moet het dier niet langer beschikbaar zijn in de Animals collectie van de Enclosure.

Requirements:

1. Wanneer een Enclosure object wordt aangemaakt, moet dit object een lege Animals collectie bevatten.
2. Een Animal object moet kunnen worden aangemaakt met een unieke Id en een Name.
3. De Enclosure klasse moet een methode of mechanisme bevatten waarmee een Animal object kan worden verwijderd uit de Animals collectie.
4. Na verwijdering moet het mogelijk zijn om te verifiëren dat het Animal object niet langer in de Animals collectie van de Enclosure aanwezig is.

Test Cases: Dier verwijderen uit Enclosure

- Beschrijving: Controleer of een Animal object succesvol kan worden verwijderd uit de Animals collectie van een Enclosure.
- Input: Verwijder uit de Animals collectie van een Enclosure.
- Verwachte Uitkomst: Het Animal object moet eerst aanwezig en daarna afwezig zijn in de Animals collectie van de Enclosure.

User story Titel: Climate Tropical

Beschrijving: Deze test controleert of de Climate property van een Enclosure object correct kan worden ingesteld op Tropical.

Acceptatiecriteria:

- Een nieuw Enclosure object moet correct worden aangemaakt.
- De Climate property van het Enclosure object moet kunnen worden ingesteld op Enclosure.ClimateType.Tropical.
- De ingestelde waarde van de Climate property moet gelijk zijn aan Enclosure.ClimateType.Tropical.

Specifiacties en requirements:

Klasse: Enclosure

- Eigenschap: Climate
    - Type: Enclosure.ClimateType
    - Mogelijke Waarden: Temperate, Tropical, Arctic, etc.

Enums:

- ClimateType: Een enumeratie die verschillende klimaten beschrijft, waaronder Tropical.

Test Cases:

- Actie: Maak een nieuw Enclosure object aan en stel de Climate property in op Enclosure.ClimateType.Tropical.

- Verwachting: De waarde van de Climate property is Enclosure.ClimateType.Tropical.

User story Titel: Check Security

Beschrijving: Deze test controleert of de Climate property van een Enclosure object correct kan worden ingesteld op Tropical.

Acceptatiecriteria:

- Een nieuw Enclosure object moet correct worden aangemaakt.
- De Climate property van het Enclosure object moet kunnen worden ingesteld op Enclosure.ClimateType.Tropical.
- De ingestelde waarde van de Climate property moet gelijk zijn aan Enclosure.ClimateType.Tropical.

Specifiacties en requirements:

Klasse: Enclosure

- Eigenschap: Climate
    - Type: Enclosure.ClimateType
    - Mogelijke Waarden: Temperate, Tropical, Arctic, etc.

Enums:

- ClimateType: Een enumeratie die verschillende klimaten beschrijft, waaronder Tropical.

Test Cases:

- Actie: Maak een nieuw Enclosure object aan en stel de Climate property in op Enclosure.ClimateType.Tropical.

- Verwachting: De waarde van de Climate property is Enclosure.ClimateType.Tropical.

HABITAT TEST

User story: Habitat test

Beschrijving: Deze testklasse HabitatTest controleert of het systeem in staat is om het klimaat- en habitattype van een dierverblijf correct te identificeren. Dit is essentieel om ervoor te zorgen dat dieren in een passende omgeving worden gehuisvest, wat bijdraagt aan hun welzijn.

Acceptatiecriteria:

- Het systeem moet het klimaattype van een verblijf correct kunnen identificeren en retourneren.
- Het systeem moet het habitattype van een verblijf correct kunnen identificeren en retourneren.

Specificaties en Requirements:

Klimaattype Identificatie:

- Het systeem moet in staat zijn om het klimaattype van een verblijf te identificeren en te retourneren als Enclosure.ClimateType.
- Beschikbare klimaattype opties: Tropical, Temperate, Arid, enz.

Habitattype Identificatie:

- Het systeem moet in staat zijn om het habitattype van een verblijf te identificeren en te retourneren als Enclosure.HabitatType.
- Beschikbare habitattype opties: Forest, Savannah, Desert, enz.

Test cases:

Klimaattype Identificatie Test

- Setup:
  - Maak een nieuwe instantie van Enclosure aan.
  - Stel het klimaattype in op Enclosure.ClimateType.Tropical.
- Actie:
  - Roep de property Climate van de instantie aan.
- Verwachting:
  - Controleer of de geretourneerde waarde gelijk is aan Enclosure.ClimateType.Tropical.

Habitattype Identificatie Test

- Setup:
  - Maak een nieuwe instantie van Enclosure aan.
  - Stel het habitattype in op Enclosure.HabitatType.Forest.
- Actie:
  - Roep de property Habitat van de instantie aan.
- Verwachting:
  - Controleer of de geretourneerde waarde gelijk is aan Enclosure.HabitatType.Forest.

INFRASTRUCTURE TESTS

User story: Database Test

Beschrijving: Deze test is gemaakt om de verificatie van de Database klassen te testen voor het toevoegen een Animal object en vervolgens op te halen. Het doel is om zeker te weten dat een dier toegevoegd kan worden aan de database, en teruggehaald kan worden met dezelfde eigenschappen.

Acceptatiecriteria:

- De test moet een Database-object en een Animal-object correct kunnen instantiëren.
- De AddAnimal-methode van de Database-klasse moet het Animal-object correct toevoegen.
- De GetAnimal-methode van de Database-klasse moet het toegevoegde Animal-object correct kunnen ophalen met behoud van de eigenschappen (zoals Name).
- De test moet slagen als het toegevoegde dier correct wordt teruggehaald met dezelfde naam.
- De test moet falen als het toegevoegde dier niet correct wordt teruggehaald.

Specificaties en Requirements:

Database Klasse:

- Methode AddAnimal(Animal animal):
  - Voegt een Animal-object toe aan de database.
- Methode GetAnimal(int id):
  - Haalt een Animal-object op uit de database op basis van het id.

Animal Klasse:

- Eigenschap Id (int):
  - Een unieke identificatie voor elk dier.
- Eigenschap Name (string):
  - De naam van het dier.

Test cases:

Setup:

- Maak een instantie van de Database-klasse.
- Maak een instantie van de Animal-klasse met Id = 1 en Name = "Lion".

Actie:

- Voeg het Animal-object toe aan de Database met de methode AddAnimal.
- Haal het Animal-object op uit de Database met de methode GetAnimal.

Verwachting:

- De naam van het opgehaalde dier (retrievedAnimal.Name) moet gelijk zijn aan "Lion".

User story: Load test

Beschrijving: De test RunLoadTest binnen het project maakt gebruik van de NBomber library om een belastingstest uit te voeren op de Database-klasse. Het doel van de test is om de prestaties van de database te evalueren bij het toevoegen en ophalen van dieren onder hoge belasting. De test bestaat uit twee stappen: het toevoegen van dieren aan de database (AddAnimal) en het ophalen van dieren uit de database (GetAnimal).

Acceptatiecriteria:

- De test moet een Database-object en een Random-object kunnen instantiëren.
- De AddAnimal-stap moet succesvol dieren aan de database kunnen toevoegen.
- De GetAnimal-stap moet succesvol dieren uit de database kunnen ophalen.
- De belastingstest moet 1000 verzoeken per seconde kunnen simuleren gedurende 5 seconden.
- De test moet rapporteren hoeveel verzoeken succesvol waren en hoeveel verzoeken faalden.

Specificatie en requirements:

- Gebruik van Step.Create om de stappen AddAnimal en GetAnimal te definiëren.
- Gebruik van ScenarioBuilder.CreateScenario om een scenario met de stappen te creëren.
- Gebruik van Simulation.InjectPerSec om de belasting van 1000 verzoeken per seconde te simuleren.

Test cases:

Setup:

- Maak een instantie van de Database-klasse.
- Maak een instantie van de Random-klasse.
- Definieer de stap AddAnimal die willekeurige dieren toevoegt aan de database.

Actie:

- Simuleer 1000 verzoeken per seconde voor de AddAnimal-stap gedurende 5 seconden.

Verwachting:

- Alle AddAnimal-verzoeken moeten succesvol zijn met de respons "Animal added to database".

GEMAAKTE KEUZES

Ook werd als feedback gegeven om een terugkoppeling te maken naar de feedback die geven is en het te verweken in het project en documentatie. Alle gemaakte keuzes en terugkoppeling zijn te vinden in dit document.

Wij hebben ons niet helemaal aan het plan gehouden. Zo hebben wij gekozen om niet verder te gaan met Azure voor het testen en deployen. Aan de hand van het lesmateriaal hebben wij gekozen om de deployment fase aan de hand van Docker te doen. Docker biedt een veel makkelijkere en betere samenwerking met Kubernetes voor het deployen en maken van build images. Ook met betrekking tot het testen hebben wij gekozen om met GitHub actions te werken voor geautomatiseerde tests. Onze code werd sowieso op GitHub zelf geplaatst voor samenwerking en publiceren. Daarom leek het ons toch een logischere keuzen om de tests met GitHub Pipelines uit te voeren. Ook werd dit in de les aangeraden en kregen wij hier meer hulp en ondersteuning bij.

Tijdens de demo presentatie werd ons feedback gegeven om aan te werken. Onder andere werd ons gevraagd om een derde infrastructuur test te maken en alle andere infrastructuren testen te omschrijven en uit te leggen waarom deze test gekozen zijn. In dit document worden de infrastructuren testen omschreven en de doelen van de testen.

Ook hebben wij de Pipeline gemaakt en de orde aangepast op advies van Tom. Zo worden eerst de testen getest en daarna de build uitgevoerd. Ook hebben wij de security en deployment test gemaakt en werkend gekregen in de pipeline. Tot slot hebben wij de badges binnen de ReadMe file aangepast en geüpdatet.