Course ID: 5364DCBS6Y

**DevOps and Cloud-based software—**

# Course project list

This list contains projects of the following types.

1. Migrate an existing application to the cloud
2. Investigate microservice application performance from an operation perspective
3. Check advanced DevOps technologies
4. Build a cloud-native application from scratch


**dr. Soveizi, Nafiseh. <n.soveizi@uva.nl>**

| | |
|---|---|
| ID | 1 |
| Title | Migrate service to the cloud |
| Contact | Nafiseh Soveizi <n.soveizi@uva.nl> |
| Description | A client wants to launch a Ticket Booking process, involving a workflow with service tasks that are currently not cloud-native (please refer to the code [1] available on Git). The client intends to transition these services to the cloud and evaluate operational costs across different cloud environments. |
| Requirements | 1. Develop a flexible and elastic online ticket booking prototype.<br>2. Implement continuous service operation with features such as monitoring, anomaly warnings, and fast response times. |
| Expected output | 1. A working cloud application<br>2. Cost estimation<br>3. Elastic cloud resource management |

| | |
|---|---|
| References | [1] https://github.com/berndruecker/ticket-booking-camunda-8 |


## dr. Siamak Farshidi (s.farshidi@uva.nl)

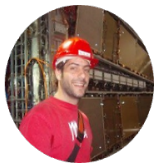

| | |
|---|---|
| ID | 2 |
| Title | Cloud-Based Event Management System |
| Contact | Siamak Farshidi (s.farshidi@uva.nl) |
| Description | The project involves developing a cloud-based event management system that allows users to create, manage, and attend events virtually. The system will use a microservices architecture, with each microservice handling a specific functionality of the application (e.g., user registration, event creation, notifications). The project includes setting up these microservices in a cloud environment, ensuring scalability and reliability. Students will also implement CI/CD pipelines for automated testing and deployment of the microservices. |
| Requirements | Microservices for different functionalities (User Management, Event Management, Notification Service).<br>Cloud deployment of microservices with scalability considerations.<br>Continuous Integration and Continuous Deployment (CI/CD) pipelines.<br>Authentication and authorization for users.<br>Database management for storing user and event data.<br>Interface for users to interact with the system. |
| Expected output | A working prototype of the cloud-based event management system.<br>Scalable and reliable microservice architecture in the cloud.<br>Automated pipelines for deployment and testing.<br>User interface for managing and attending events.<br>Documentation of the system architecture, CI/CD setup, and deployment strategies.<br>Analysis of cloud resource usage and cost estimation. |
| References | Microservices Architecture Design |

| | |
|---|---|
| | CI/CD with Jenkins, Docker, Kubernetes |

## Dr. Gabriel Pelouze <gabriel.pelouze@lifewatch.eu>

| | |
|---|---|
| ID | 3 |
| Title | migrate a classical application to the cloud |
| Contact | g.pelouze@uva.nl |
| Description | A consortium of European Research Infrastructures has developed a knowledge base to aggregate research assets from many organizations. These assets have different types (eg. documents, datasets, services, computational notebooks), and origins and are updated with different frequencies. The consortium has developed an indexing pipeline [1] that gathers assets metadata from different sources, converts them to a standard schema, and stores them in an Elasticsearch database. This supports a search engine [2] that allows scientific community members to search across millions of research assets. The indexing pipeline is currently written as a monolithic Python application containerized in Docker. The list of resources to index (data repositories, search keywords, queries, etc.) is bundled with the code. |
| Requirements | 1. An indexing pipeline that can ingest documents from different sources and store them in a common Elasticsearch instance [1]<br>2. A framework that allows system administrators to<br>a. run the indexing pipeline for specific sources or resource types<br>b. periodically run indexing tasks<br>c. observe the indexing status<br>3. A multi-cloud solution that allows the consortium to avoid vendor lock-in. A possible solution is to use Kubernetes.<br>4. A system allowing administrators to update the list of resources to index and automatically trigger reindexing upon update. |
| Expected output | Minimally:<br>1. Design of the created solution |

| | |
|---|---|
| | 2. A working prototype of the framework described in req. #2<br>3. A DevOps pipeline supporting build, testing, releasing, and deployment to multi-cloud<br>4. Monitoring system (can be removed if scope too broad) |
| References | [1] https://github.com/QCDIS/kb-indexer<br>[2] https://search.envri.eu |



## dr. Spiros Koulouzis <S.Koulouzis@uva.nl>

| | |
|---|---|
| ID | 4 |
| Title | Dynamic Allocation of Cloud Resources with IaaC |
| Contact | S.Koulouzis@uva.nl |
| Description | The development of sophisticated cloud-based virtual research environments aims to facilitate advanced data-intensive applications, including those in environmental and earth sciences (e.g. ENVRI and LifeWatch) and the medical field (e.g., CLARIFY).<br>A crucial part of these applications is the execution of container-based workflows run by a workflow engine such as Argo deployed on a Kubernetes cluster. However, the workflow engine is deployed on a Kubernetes cluster using static infrastructure private cloud. |
| Requirements | 1. Using an IaaC open source, cross-platform tool (e.g. Ansible, AWX, Terraform), we want to allocate and release compute resources in AWS.<br>2. The solution should be multi-cloud, i.e., not dependent on a specific cloud technology or provider.<br>3. Connect with monitoring to provide elastic provisioning of cloud resources. |
| Expected output | 1. Minimally: A working prototype as described in requirement 1 that includes a Git repository with all the relevant files (scripts, YAML configurations, README.md, etc.).<br>2. For extra credit: Include a solution for one more cloud provider/technology e.g. OpenStack.<br>3. For extra credit: Allow for automated elastic provisioning of cloud resources based on custom metrics. |
| References | https://devopscube.com/infrastructure-as-code-configuration-management/ , https://www.youtube.com/watch?v=zWw2wuiKd5o |

**dr. Peide Zhu <peide.zhu@lifewatch.eu>**

| | |
|---|---|
| ID | 5 |
| Title | Migrate service to the cloud |
| Contact | Peide Zhu, peide.zhu@lifewatch.eu |
| Description | SearchX is a scalable collaborative search system to facilitate collaborative search and sensemaking. It includes the frontend that serves the interface and the backend that is responsible for fetching search requests to the search provider and managing the application's data. A university wants to migrate the SearchX system to the cloud. The solution should require minimal maintenance for the engineers who manage the multi-cloud environments. Currently, the application is deployed as a block, including a frontend, backend, and several dependencies (MongoDB, Redis, and Elasticsearch). They should be run as separate services independent of the front-end and back-end. |
| Requirements | Minimally:<br>1. Separate the Mongodb/Redis/Elasticsearch from the backend and frontend<br>2. A pipeline to containerize the application components, test them, and publish the images. This pipeline should be triggered by updates to the code repository.<br>3. Automated deployment of the application components to a multi-cloud environment (this can rely, for example, on Kubernetes)<br>4. All code must be void of plaintext secrets<br>For extra points:<br>5. Monitoring<br>6. Automated rollback on errors (e.g. using canary testing) |
| Expected output | 1. SearchX front-end and back-end services running on the cloud.<br>2. Automatic Mongodb/Redis/Elasticsearch service monitoring/ management. |
| References | https://github.com/searchx-framework/searchx-frontend<br>https://github.com/zpeide/searchx-backend-aqgsal<br>https://www.elastic.co/guide/en/elasticsearch/reference/7.17/starting-elasticsearch.html<br>https://www.mongodb.com/docs/v6.0/tutorial/install-mongodb-on-ubuntu/ |

**Maurice Mouw <maurice.mouw@sue.nl>**

| | |
|---|---|
| ID | 6a |
| Title | Migrate a classical application to the cloud |
| Contact | Maurice Mouw |
| Description | A university wants to migrate its attendance system to the cloud. The solution should require minimal maintenance for the engineers who manage the AWS cloud environment as they already have a staff shortage. One of the engineers suggested a serverless solution to reduce infrastructure maintenance to limit additional work on infrastructure for the IT department. |
| Requirements | 1. An attendance registration system prototype with back and front-end<br>2. Scalability to handle peak traffic<br>3. Automated deployment of back and front-end<br>4. All code must be void of plaintext secrets |
| Expected output | 1. Design of the created solution.<br>2. Working attendance registration system.<br>3. A system that scales out during peak traffic.<br>4. A solution deployed via DevOps best practices<br>a. Documentation is generated<br>b. Deployed via pipeline(s)<br>c. Secret management is applied |
| References | Example: https://github.com/Ubaid-Manzoor/AttendanceApp-BackEnd/blob/master/version_II/app/Services/userServices.py |

## Nathan Keyaerts <nathan.keyaerts@sue.nl>

| | |
|---|---|
| ID | 7a |
| Title | Online Student Portal |
| Contact | Nathan Keyaerts <nathan.keyaerts@sue.nl> |
| Description | The ‘We learn a lot’ university department seeks to create a collaborative online portal where students can share and access solutions to exercises, study summaries, and other academic resources. The platform should encourage knowledge sharing, provide content organization by subject or course, and include mechanisms for content quality validation. |
| Requirements | Starting from scratch build:<br>1. User authentication and role-based access control.<br>2. Organized content repository categorised by courses and topics.<br>3. Content validation system, allowing peer review.<br>4. Search functionality with filters for easy content discovery.<br>5. Notification system for updates or new content in subscribed categories. |
| Expected output | 1. A secure, scalable, and user-friendly student collaboration portal deployed in the cloud.<br>2. Content management system allowing for easy upload, categorization, and retrieval of study materials.<br>3. Documentation on system architecture, security measures, and a cost estimate for varying user loads.<br>Extra:<br>4. Analytics dashboard showing popular content, active contributors, and user engagement metrics. |
| References | Example: https://github.com/ivanscorral/secure-file-sharing-backend |

| | |
|---|---|
| ID | 7b |
| Title | Carpooling Application |
| Contact | Nathan Keyaerts |
| Description | An environmental initiative, ‘Mother Nature’s car,’ aims to reduce carbon emissions by facilitating carpooling among travelers. The application should allow users to offer or join rides, track shared kilometers, and rate other users. |
| Requirements | 1. Secure user profiles with preferences for ride-sharing (e.g., gender, smoking, vehicle type).<br>2. Real-time scheduling of rides.<br>3. Environmental impact tracker showing CO2 savings and other metrics.<br>4. Feedback and rating system for drivers and passengers. |

| | |
|---|---|
| Expected output | 1. A scalable and secure ride-sharing application.<br>2. Reports on environmental impact, user engagement, and system usage.<br>3. Financial analysis of the cost-effectiveness of the system. |
| References | Example:<br>https://github.com/amitshekhariitbhu/ridesharing-uber-lyft-app |