**Mattias Skog, Tero Ala-Hulkko, Farshid Mohebbi, Ahmed Zafar**

**VOLUNTEER 365 – VOLUNTEER MANAGEMENT**

**Project Development Strategy and Integration**

**Written assignment**

**CENTRIA UNIVERSITY OF APPLIED SCIENCES**

**Cloud-Based Software Engineering**

**March 2024**

**CONCEPT DEFINITIONS – Do we need this?**

**CONCEPT DEFINITIONS**

**CONTENTS**

[1 INTRODUCTION 1](#_Toc164837570)

[2 PROJECT KICKOFF REFLECTION 2](#_Toc164837571)

[3 ARCHITECTURE 4](#_Toc164837572)

[3.1 Computing 4](#_Toc164837573)

[3.2 Database 5](#_Toc164837574)

[3.3 Network 6](#_Toc164837575)

[3.4 Security 7](#_Toc164837576)

[4 CONCEPT INTEGRATION 9](#_Toc164837577)

[4.1 Caching 9](#_Toc164837578)

[4.1.1 Benefits of caching 9](#_Toc164837579)

[4.1.2 Challenges related to caching 10](#_Toc164837580)

[4.1.3 Implementing caching ~~in Volunteer management~~ 10](#_Toc164837581)

[4.2 Event-Driven Architecture 11](#_Toc164837582)

[4.2.1 Advantages and disadvantages 12](#_Toc164837583)

[4.2.2 Implementing event-driven architecture 12](#_Toc164837584)

[4.3 CI/CD practices 14](#_Toc164837585)

[4.3.1 Challenges related to CI/CD 15](#_Toc164837586)

[4.3.2 Implementing CI/CD practices 15](#_Toc164837587)

[5 TECHNICAL PLANNING 17](#_Toc164837588)

[6 LEVERAGING LEARNING FOR PROJECT ENHANCEMENT 19](#_Toc164837589)

[7 PREPARATION FOR FINAL MEETING 21](#_Toc164837590)

[8 SETTING GOALS AND ACCOUNTABILITY 22](#_Toc164837591)

**REFERENCES 1X**

**APPENDICES**

**FIGURES**

FIGURE 1. Success factors for competitive edge 4

**PICTURES**

PICTURE 1. Motorcycling in Finland´s Winter 5

PICTURE 2. Heading 5

**TABLES**

TABLE 1. Age distribution of the respondents 5

TABLE 2. Heading 5

# INTRODUCTION

Whenever working on a project, it is important to have a good strategy and alignments goals, otherwise you might find yourself going around in circles or working on the wrong things. In this report we will have a look at how we will approach our term project, how we keep each other accountable and what tools we will be using. We will also discuss how we intend to integrate advanced concepts like caching, event-driven architecture, and CI/CD practices into our project.

For all project members, this was our first foray into AWS services. As such the project has evolved over the duration of the course. As we learned more complex features or noticed something was missing from what we had, we aimed to implement them to our project. This could not have been possible without intense collaboration and engagement from our project members.

The term project offered us an opportunity to make use of several skills learned during our master’s education. Especially the Modern software architectures, Agile development methods, and Leadership of software projects courses have shaped how our architecture and project management strategies turned out.

# PROJECT KICKOFF REFLECTION

* *Summarize the objectives, roles, and alignment goals your team has established for your project.*
* *Include a figure or diagram that represents your team's initial project kickoff strategy. Reference the "Initiating Your Project Journey" slide for inspiration.*

Our project was started during class session on March 6th, 2024. During that class session most of the time was spent brainstorming around possible topics for our term project. We came up with a few good ideas but still couldn’t decide which idea to go for. Due to that we continued our discussions after class and finally decided that we would focus on a volunteer management system that we decided to call Volunteer 365. In other words, a web portal where event organizers can add events for which they need volunteers. Anyone can visit the portal, browse events and apply as volunteers at the events. Our project is explained in more detail in the first written assignment of the course. That report also contains an initial architectural design for the project.

During our online meetings between the classes, once we had decided on a topic, we specified our project objectives and assigned roles to all team members according to the following:

* Mattias Skog – Project Manager
* Tero-Ala Hulkko – Cloud Architect
* Farshid Mohebbi – Project Engineer (focus on application development and testing)
* Ahmed Zafar – Project Engineer (focus on security and application testing)

When deciding on roles within the team, we started by discussing each team members background, interests and experience, and to clarify strengths and weaknesses, in order to be able to assign each team members with an appropriate role. No one in our team really had experience from project management but Mattias was appointed project manager as he had already been responsible for organizing and leading the online meetings and keeping everyone up to date on the progress. Unfortunately, everyone was not able to join all sessions but we recorded, and transcribed, every session so it would be easy to catch up later. Tero will work as our main cloud architect will also be responsible for the setup of the AWS environment. Farshid will be responsible for security in our AWS environment and Ahmed will be responsible for the development of the app. As we have a small project team everyone will have to help wherever help is needed. In addition to their primary tasks, Mattias and Tero will also be helping with development of the application and Farshid will be responsible for testing and quality assurance.

During class session on March 11th, 2024, we continued our discussion around our term project, and this was the sessions where our project was officially initiated. We started the session with discussing and reviewing what we had discussed during previous session and during our online meeting. After reviewing the outcomes of our previous meeting we moved on to discuss about any possible issues or challenges we are facing at the moment. As we now had learned more about AWS from the AWS Academy online courses and during today’s class, we realized that have might have some flaws in our architecture and that there are things we need to reconsider. Our first intention was to deploy our solution using Elastic Beanstalk but we had misunderstood the concept of Elastic Beanstalk so we needed to change our architecture a little. The refreshed architecture is presented in chapter 3 Architecture. During the class session on March 11th we also defined initial milestones for our project and immediate next actions for our project. The milestones are presented in chapter 5 Technical Planning and the immediate next actions are presented in chapter 8 Setting Goals and Accountability. Before wrapping up the meeting and deciding on the date for the next meeting, as it’s important to confirm understanding and commitment of all team members, we ensured that everyone knows what we are building, what the next steps are and what it is required from each team member.

A diagram of a flowchart

Description automatically generated

FIGURE 1. Project kickoff strategy

# ARCHITECTURE

As mentioned in chapter 2 Project Kickoff Reflection we had to change our architecture due to misunderstandings. The new architecture is presented in the diagram below.



FIGURE 2. The refreshed architecture.

## Computing

Our goal was to make a highly available, fault tolerant and scalable computing strategy for our application. Our architecture makes use of many of the AWS well-architected frameworks design principles such as not having to guess capacity, horizontal scaling, and automatic failure recovery (AWS Well Architected Framework 2024).

At the heart of our computing strategy are EC2 instances. We chose EC2 as our processing method because of its flexibility. By utilizing regular EC2 instances instead of for example elastic beanstalk, we do not limit our options in the future. This choice comes at the expense of added difficulty, as we need to define the instance environments ourselves. (AWS Academy Cloud Architecting, Module 4 Student Guide 2022, 11-12)

Our architecture consists of 2 computing tiers. Business logic and webserver are separated into different EC2 instances. Together with the database tier, this makes our architecture a 3-ter system. Separating architecture into multiple tiers offers benefits such as improved scalability as we only must scale the parts we need. Added benefits come from improved security as accessing our data and logical tiers becomes ever more difficult.

We chose the method of scaling computing infrastructure to be auto scaling groups. Auto scaling groups will be set up to automatically scale our EC2 instances horizontally. This works particularly well in our environment that utilizes multiple availability zones. All availability zones will be scaled in or out concurrently as needed. To trigger scaling inward and outward scaling, CloudWatch will be used to monitor the utilization of EC2 resources. This ensures that there will be sufficient resources to meet demand and on the other hand, resources are not being wasted during quiet periods.

Elastic load balancers are the final piece of our computing strategy. ELB’s are a part of an AWS region. They can route requests to EC2 instances that have available processing power. ELB’s help us avoid cases where problems in one availability zone hinders performance. Traffic can be routed to a working EC2 instance safely. ELB’s also work well in web applications that need to save session data.

## Database

When designing the data storage system we wanted to focus on speed, availability, and reliability. Designing databases is its own skill altogether and when we were discussing our prior experience, knowledge of SQL databases was thorough. Despite some advantages a no- SQL database could provide in speed, the well-structured SQL database along with expertise in the field lead us to choose RDS as our AWS service of choice (AWS Academy Cloud Architecting, Module 5 Student Guide 2022, 17).

The structured nature of an SQL database works well for our application. Our application stores user, organization, and event data that is uniform. All events, organizations and users need to be interconnected which can lead into somewhat complex queries. SQL databases work well for complex queries when a no-SQL counterpart could have some loss of performance.

Another key reasoning for choosing RDS is that it is a fully managed service. This means that AWS can take care of as many administrative tasks as we please. Tasks such as software patching and provisioning can be managed by AWS. (AWS Academy Cloud Architecting, Module 5 Student Guide 2022, 21-22).

Our database infrastructure consists of two AWS RDS instances. One of the instances is the primary instance, and the other one is a second instance, which replicates the data from the primary. The second instance is the backup in case the primary becomes unavailable. The backup instance is in a different availability zone to maximize fault tolerance. It can also be used as a read only instance, lightening the load for the primary. This system makes our data very durable.

Alongside RDS instances our database structure makes use of ElastiCache for Redis. ElastiCache for Redis is a caching tool that can be used to make cached database queries significantly faster (AWS Academy Cloud Architecting, Module 11 Student Guide 2022, 53). We plan to implement is the Lazy loading caching strategy. In this method, our cache only caches data that is needed. Lazy loading is the preferred method of caching for social media applications.

## Network

Our networking needs to make it easy and efficient access to our website. While accessing our website needs to be highly available, it also needs to be able to keep intruders out.

We use the Route 53 service to make our website discoverable online. One benefit Route 53 offers is the ability to register domain names and use them for our website. Additionally, Route 53 allows us to set up DNS routing. This can be helpful in case we decide to go global with our application. People from different parts of the world can be routed to different versions of the website. (AWS Academy Cloud Architecting, Module 9 Student Guide 2022, 49) This allows us to offer a more personalized experience for our users.

The second service we use that does not reside within our VPC, is the AWS CloudFront. CloudFront is a mixed bag of benefits in all areas of the application. The main benefits in the first phase of development are reduced costs and increased performance, thanks to its caching capability. Several user generated queries can be quickly responded to without the need to query our actual databases. Further benefits come from CloudFront’s ability to use edge locations. Users can be served closed to their location, reducing latency. Another benefit that has potential to be a life saver in the current geopolitical environment is the DDoS protection that can be enabled with CloudFront. (AWS Academy Cloud Architecting, Module 11 Student Guide 2022, 22-23)

Communication to and from our VPC goes through an internet gateway. Our VPC is split into multiple components for increased availability and security. Within the VPC, traffic is routed to each availability zone by using Elastic Load balancers and Auto-scaling groups.

Our application makes use of redundancy by residing in multiple availability zones. This is to ensure functionality of our application in a scenario where an entire availability zone goes down.

In our application, EC2 instances and database instances reside in private subnets. This allows us to further limit access to the instances from outside of our VPC. SSH can still be used to complete any maintenance tasks EC2 instances may require.

Instances inside of our private subnets can only send messages to the internet through public subnets. Our public subnets contain only a NAT gateway, that is used to send messages back to the internet and our users.

## Security

Online security has become a very diverse topic. It can split into multiple angles such as keeping away intruders, making sure data is not compromised, and even legislation. We must be able to protect our physical infrastructure, data, and users. Security is more important than ever, and we wanted our architecture to reflect that.

Several security features have been discussed already in this document, such as DDoS protection and private subnets. The following features are implemented with only security in mind.

We use features that AWS offers to make sure our servers remain untampered with. Our network implements ACLs to restrict traffic into a specific type. In our case this would be to only allow TCP traffic into our subnets. ACLs help us avoid giving access to malicious entities and if they gain access, ACLs limit what they can accomplish while inside our network.

Another method of limiting access to our VPC is the use of security groups. Security groups allow a more granular control of traffic to specific instances within our network. They can be used to create rules that allow for example a specific IP address to connect via PuTTy to do maintenance on EC2 instances.

As we are an organization, several people will need to have access to our AWS management console. This can create vulnerabilities as each person has different skillsets and expertise. One way to avoid mistakes that can be made with good intentions in mind is to use IAM roles. IAM roles can limit access to which parts of the architecture each person can modify. Principle of least privilege is applied making sure accessors only have access to what they need. Additionally, for IAM users, we enforce multi-factor authentication. While we use IaC to manage our environment, IAM roles should still be in place. Another benefit of IAM roles is it can be used to limit AWS services access rights. This can be used to avoid any unintended interactions that an EC2 instance, for example, tries to do. (AWS Academy Cloud Architecting, Module 8 Student Guide 2022)

We must also be able protect users and their accounts. We use Cognito service to connect users to their accounts. Cognito is a user management – service that grants a temporary privilege for users to access our application in the AWS environment.

Cognito is the AWS way to leverage user pools and identity pools. These can help simplify the user creation process from the developer’s standpoint. Cognito supports signing in using Google, Facebook and Apple accounts which makes it a viable solution. Additional security comes from the support of multi-factor authentication. This is not something we are mandating from the users, at least not at first.

Using Cognito, user information is encrypted. (AWS 2024) Encryption is a vital part of data security. In addition to Cognito, our architecture leverages encryption in two ways. First is to use HTTPS protocol when communicating with a user’s browser. HTPPS encrypts data in transit, which means that even if captured mid transit, it cannot be read. Another encryption resides in our RDS. RDS database data is encrypted at rest to protect it from prying eyes. Proper encryption is required by several government agencies.

Since our application handles user data, under the EU laws, users have several rights pertaining to them. The data we handle can be very sensitive as at some point, we may need to verify a user’s identity. This can arise if event organisers demand to know exactly who they are enlisting

from our application. GDPR is a vast collection of rules that govern data privacy. We must be able to implement notifying users about the information we collect, right of access to your data, right of erasing the data, and many more rights. (gdpr.eu)

With all these security features implemented, we will have a good starting point for the security element. We must be able to set up monitoring and logging for many parts of the application to know if a breach has occurred. If a malware manages to attach itself and monitor or steal information, we must be able to detect it and figure out how it got there. Security is an ever-changing landscape that must be maintained diligently.

# CONCEPT INTEGRATION

* *Detail how your team plans to integrate at least two of the advanced concepts (caching, event-driven architectures, CI/CD practices) into your project.*
* *For each concept, describe the expected benefits and any challenges you anticipate.*
* *Create or include a conceptual diagram that illustrates how one of these concepts will be implemented within your project structure.*

As our project needs to implement some advanced concepts like caching, event-driven architecture or CI/CD practices we discussed these concepts during the project session during our second class. We discussed how these concepts could be implemented, if we would have any benefits from them, possible challenges they could cause etc. The following chapters describe in more detail how we plan to integrate these concepts into our project.

## Caching

Retrieving data, from a database or an S3 bucket for example, can be costly if it’s done for lots of users constantly. One way to reduce the need for data retrieval is to implement caching.

### Benefits of caching

Caching in computing means temporarily storing frequently accessed data in a more accessible location. There are two main benefits that caching offers for our application: speed and cost. (AWS Academy Cloud Architecting, Module 11 Student Guide 2022, 11)

Cache is a high-speed data storage layer. Reading from a cache is typically much faster than querying a relational database, which our application uses. Cache is also physically located closer to the user than the backend database. These properties of caching make retrieving data much faster. The speed allows us to provide users with a more responsive experience. (AWS Academy Cloud Architecting, Module 11 Student Guide 2022, 11-13)

Some cost savings can be gained by using caching. AWS CloudFront is a more efficient way of retrieving data than going all the way to the EC2 instance and retrieving what is needed. Some in memory caching types however may be more expensive than simply reading from the database. However, since caching provides other features as well, such as session management, that we would need to implement anyway, the benefits may outweigh the increased cost.

### Challenges related to caching

Stale data is a drawback of caching. It means that the information presented to the user, might have

changed after it was cached. In our application, this could mean that if an event organizer changes the time for an event, cached data displayed to the user might not be correct data. (AWS Academy Cloud Architecting, Module 11 Student Guide 2022, 14)

Another challenge in caching is whether it is useful in an application that relies heavily on search results. Our application allows users to browse and search events and organizers. Since the data expected user to event organizer ratio is expected to be heavily slanted towards users, we expect caching to still provide benefits. This is because the same events and organizations will be loaded repeatedly. (AWS Academy Cloud Architecting, Module 11 Student Guide 2022, 14)

### Implementing caching

Our implementation runs the website in EC2 instances. This lets us make use of the Content delivery network, CDN. With AWS CloudFront, content like images and web components can be stored at an edge location. With CloudFront, content can be retrieved from an edge location instead of the origin server, which might exist much further away geographically. CloudFront is greatly customizable. For CloudFront to operate, a distribution must be configured. The distribution tells CloudFront the origin of the content to be cached, which content is cached and how long content will be cached, among other things.



FIGURE 3. Caching with CloudFront shortens the distance from user to data.

Another caching strategy that needs to be implemented is storing user session information. Storing user sessions reduces the need to validate logins repeatedly. This offers a much-improved user experience as well as reduced workload for validation.

Since our application uses a relational database, Amazon ElastiCache for Redis can be used to store session information. Redis is an in-memory data store with high performance. ElastiCache supports lazy loading, which is perfect for a social media application. A Redis cluster can be launched using the ElastiCache console on AWS.



FIGURE 4. Lazy loading only queries the database after a cache miss. Information will be written to cache for later use.

The downside of Redis is that it requires lots of RAM to function. This can lead to high costs. Alternative to Redis is using sticky sessions. Sticky sessions is a feature of the Elastic LoadBalancing Load Balancer. ELB can direct traffic to the instance that is managing a user’s session. This can lead to lost sessions when instances are terminated.

## Event-Driven Architecture

In an event-driven architecture is an integration model and design pattern that supports decoupled applications, for example microservices. In an event-driven architecture events are identified and used to communicate and trigger actions between applications, or even withing the same application. For example, when something happens in module A of an application, an event is triggered, and the event is published through a message broker or service bus. Modules B and C of the application subscribe to, or listen for, this event and then performs some action based on the published event from module A. AWS provides many out of the box services that helps implement event-driven architecture. Among them AWS Lambda, Amazon SNS (Simple Notification Service), Amazon SQS (Simple Queue Service), Amazon EventBridge and Amazon MQ.

### Advantages and disadvantages

One of the biggest advantages of event-driven architecture is that it promotes loose coupling of applications. This means that applications can easily be split into several smaller independent and autonomous applications, a.k.a. microservices, each with a specific purpose, developed by different teams and using different programming languages. As the applications usually communicate asynchronously by publishing and subscribing to events using a service bus or a message broker, the nature of these decoupled applications enables a high level of flexibility when adding new functionality. The publishers of events do not need to know who the receivers are, and the receivers do not need to know who the publishers are, they only need to know what kind of events are being published. In addition, applications that are built using event-driven architecture are often highly scalable and extensible.

Even if even-driven architectures have many benefits they also come with disadvantages. An event-driven architecture is usually much more complex than traditional architecture. This also makes it much more difficult to test, troubleshoot and monitor the application.

### Implementing event-driven architecture

Our Volunteer 365 application could be implemented using event-driven architecture as it’s easy to identify events that could be used to trigger actions and other events. Here are only a few examples:

* A new user registers.
* A new event is created.
* A user applies for volunteering at an event.
* An application is accepted or rejected.
* A user subscribes to a newsletter.
* A user updates his notification settings.

Despite that it’s easy to identify events in our application, and even if it would add flexibility, we have decided not to implement the first version of our application using an event-driven architecture. This is because of the complexity and development time that it adds. However, we will use Amazon SNS to send out notifications by e-mail to users of the application when new events are created, users apply for volunteering, applications are accepted or rejected, user notification settings are updated etc.

Amazon SNS has a well-documented and easy to use API that will be called by our application when certain events occur. To send out notifications using Amazon SNS a so-called topic must first be created in the system. A topic defines how the messages are sent and to whom. When you want to send a message using a topic, all you must do is to publish your message to the given topic and the message will be sent out to all subscribers of the topic.

A diagram of a diagram

Description automatically generated

FIGURE 5. Creating an organization also creates a topic in Amazon SNS.

By default, a topic will be created for every organization that is added to Volunteer 365 so that users can subscribe to events from their favorite organizers. When a logged in user looks up an organizer and clicks subscribe, the user’s e-mail address will be added to the organization specific topic in Amazon SNS. Later, when the given organization creates a new event, a notification will be sent out to all subscribers of that organization using the organization specific topic.

A diagram of a diagram of a globe and a house

Description automatically generated

FIGURE 6. When a user creates a subscription his e-mail address will be added to the corresponding topic in Amazon SNS.

In addition, every event category in Volunteer 365 will have its own corresponding topic. Again, if a user wants to receive notifications when events of a certain category are added to the event list, the user can subscribe to that category and by doing so the user’s email address is added to the topic. Like with organization specific events, when an event in the given category is created, a notification will be sent out to all subscribers of that category using the category specific topic. This is an easy way to manage subscriptions and it would even be possible to create n topic in Amazon SNS for each event in Volunteer 365 and this would make it possible for users to get notified about updates to the events they have chosen to follow. Amazon SNS topic can also be used to send out notifications to email addresses that are not registered as subscribers and this feature will be used to notify users when their applications have been accepted or rejected, and, also to notify organizations about new volunteering applications.

A diagram of a diagram

Description automatically generated

FIGURE 7. Creating an event publishes a message to a organization specific topic and all subscribers of the topic are notified about the new event.

## CI/CD practices

Maintaining a software environment is a lot of work. Manually updating code, software architecture, and resources takes time and is prone to causing errors. (AWS Academy Cloud Architecting, Module 10 Student Guide 2022, 10-11) We have identified some effective CI/CD practices that can help us overcome these issues.

### Challenges related to CI/CD

A big challenge for our team is our lack of experience in DevOps engineering. Learning about all the different pipelines, automation strategies and how they integrate with version control systems and AWS can easily cause information overflow. There are tons of completely different CI/CD approaches. What we must do is select a few of the best ways that help us in developing the application.

One challenge with IaC in AWS is that not all services are fully supported by CloudFormation. Some new features added to services may not be available for CloudFormation (Fowler 2020). This could create issues where the template does not fully represent the architecture.

### Implementing CI/CD practices

To make full use of CloudFormation and its many benefits, we need a code representation of our infrastructure. There are 2 quick ways to get started with using CloudFormation. One way is to use ready templates created by AWS or 3rd party vendors. These can provide a starting point for architecture that can be modified to fit our needs. (AWS Academy Cloud Architecting, Module 10 Student Guide 2022, 14-17) The second option is to use CloudFormer tool to create the IaC code from our existing environment. CloudFormer is no longer supported by AWS. Popular 3rd party tools exist to replace CloudFormer, such as Former 2, an open-source tool that scans your AWS environment and generates IaC code from it.

Tools may be helpful to learn by doing but our choice is to use the designer provided in CloudFormation. Yaml editor is also helpful as we become experts on the subject. Any architecture code and change sets will be saved in a GitHub repository for version control and storing. This yaml is then used to create stacks and deploy our environment.

Deploying both CloudFormation yaml and webserver happens through pipelines that link our GitHub and AWS resources. GitHub integration is possible with the AWS CodeDeploy service. A secure connection will be established to allow GitHub to trigger actions in the AWS environment.

In practice, pushing new code into GitHub is the first step in the pipeline’s workflow. The code will then be automatically built and after successfully authenticating with AWS using OIC D , the new version of our program will be uploaded to an S 3 bucket. From there, AWS CodeDeploy can deploy the software to our EC2 instances.



FIGURE 8. AWS CodeDeploy deploys code directly from GitHub to auto scaling groups.

To establish a connection between GitHub and AWS, IAM role can be used. By specifying the Amazon resource of the IAM role, the process can be defined to have all the rights it needs to complete the task. In addition to the IAM role, during the stack creation a GitHub repository name must be specified. This can be achieved by using custom parameters in the CloudFormation template. IAM role in combination with GitHub secrets that are used to handle integrations, security will be high. (Biradar, Moolya, 2020)

Build instructions for the webserver code and a specific S3 bucket are specified in GitHub. The S3 bucket will be used to deliver deployment information to AWS CodeDeploy. Furthermore, the yml needed to deploy on EC2 instances should be in the repository. This can include scripts that need to be run on various stages of the deployment. (Biradar, Moolya, 2020)

After manually verifying the connectivity works as intended, the process can be made fully automatic. In a fully automated pipeline, code committed to Git is immediately propagated towards AWS. (Biradar, Moolya, 2020)

# TECHNICAL PLANNING

* *Outline the technical milestones and tasks identified by your team. Discuss the delegation of responsibilities and how this planning translates your strategy into actionable steps.*
* *Include a timeline or roadmap diagram that showcases the planned milestones and their corresponding deadlines.*

From the beginning, it was clear to us that our project needs to comply with the pillars of the AWS well-architected framework. By keeping this in mind, we started our technical planning by brainstorming around our project to identify the technical requirements. At this point we identified two bigger wholes, namely the application itself and the infrastructure needed to run it in the cloud. After some further brainstorming around these two wholes, we had a clear picture of what it would require technically to implement our application. After that it was easy to define milestones for our project. We ended up defining the following high level technical milestones:

* Project start 6.3.2024
* Project initiation 11.3.2024
* Project plan done 28.3.024
* Development kickoff 2.4.2024
* Infrastructure and services setup 26.4.2024
* Application prototype 30.4.2024
* Application features finalized 17.5.2024
* Security and quality assurance 22.5.2024
* Application launch 24.5.2024
* Project assessment 30.5.2024
* Project end 31.5.2024

A graph with numbers and letters

Description automatically generated

FIGURE 9. Project milestones.

After the milestones had been defined, we looked thoroughly at what would be needed to achieve each milestones. Each milestone was broken down into smaller tasks that would help us achieve our goals. All in all, we ended up defining X tasks.

When all the tasks had been defined, we prioritized, set deadlines that were in line with the milestones and assigned the tasks. When assigning tasks the team member roles presented in chapter 2 Project Kickoff Reflection were considered. This meant tasks related to management were assigned to Mattias, tasks related to architecture and environment setup to Tero, tasks related to application development to Ahmed and tasks related to security and testing to Farshid.

More information on the tasks, their deadlines and their owners can be found on our project’s Trello board. In Trello the tasks have also been grouped (tagged) based on which milestone, application feature or infrastructure part they belong to.

A screenshot of a computer

Description automatically generated

FIGURE 10. Volunteer 365 Trello board. Not final…!!!

# LEVERAGING LEARNING FOR PROJECT ENHANCEMENT

* *Reflect on how the session’s insights can enhance your project, specifically in terms of architecture scalability, security, or deployment strategies.*
* *Propose a new feature or module for your project influenced by these insights and justify its inclusion.*

As already mentioned in chapter 2 Project Kickoff Reflection of this report, the insights of the sessions and the online AWS courses made us realize that we had made a mistake in our initial architecture. This led to a change in our architecture that is explained more in detail in chapter 3 Architecture of this report. We also learned the importance of scalability, so we have also implemented auto-scaling groups in our architecture. Scaling is also explained more in detail in chapter 3 Architecture.

As we discussed architecture scalability and deployment strategies during our session, we realized that our application and its architecture, as it was presented in the first report, might not be the best way to go. In our initial architecture diagram that was presented in the first report our application was planned to be deployed as a 2-ties application. In other words, it would be deployed on two servers, the web application on one server and the database on another. Clearly spoken, it is a monolith. Considering performance this setup is good but it's not that scalable and whenever you make a small change to the UI or the business logic, the whole application would have to be redeployed. Considering this we decided to add one tier to our application and deploy the business logic separately on an application server. After the change we would have a 3-tier architecture with a presentation tier (UI) deployed on a web server, an application/business tier deployed on a separate application server and a data tier (database) deployed on a third server. In this way we can work on, and deploy, the UI and business logic separately from each other. The application also becomes more scalable and, in the future, if we want to develop a mobile application for our volunteer management system, we can just add an API gateway that talks to our business logic. The mobile application would in turn talk to the API gateway.

Making architectural changes, or adding new features or modules, to an application always need to be evaluated carefully. If we make a change or add something it should also add more value to the product. We consider that our change in the architecture is justified as it adds value to our product. It might not be a visible change that the end user notices when they use the application, but it makes it easier to develop and deploy the application and makes it more scalable.

A diagram of a process

Description automatically generatedA diagram of a process

Description automatically generated

FIGURE 11. Evaluating and adding a new feature

# PREPARATION FOR FINAL MEETING

* *Prepare a brief outline for your team's presentation for the final meeting. This should include key points on project summary, progress highlights, challenges, and learnings.*
* *Sketch a preliminary slide layout or storyboard that highlights how you intend to present your project’s development and integration of new concepts.*

During our online meetings we have been preparing a presentation for the final meeting. The presentation will be structured in the following way:

1. Project description
2. Project team and roles
3. Architecture
   1. Diagram
   2. Basic concepts (computing, load balancing, databases, DNS, authentication, backup and disaster recovery)
   3. Advanced concepts (caching, CI/CD, event-driven architecture)
4. Technical planning and milestones
5. Accountability and tools
6. Challenges

The presentation slides have been attached to this report (APPENDIX 1).

# SETTING GOALS AND ACCOUNTABILITY

* *List the clear, achievable goals set by your team for the near future. Explain how you plan to keep each other accountable.*
* *Propose a mechanism or tool your team will use for regular check-ins and progress tracking.*

Setting goals and keeping everyone up to date on the project progress is important. A few goals for the near future that we set during our project session were:

* **Finish AWS Academy online courses** - As we are new to AWS, everyone in the team should finish the AWS Academy online courses as they give a good basis for working with AWS and its services. Everyone should complete the courses by first week of April.
* **Finalize project plan** - As a project manager, Mattias should work on a project plan and an initial timeline for the project within the coming week. Mattias will also look into if the project would benefit from an event-driven architecture.
* **Finalize architecture** - Tero will finalize the architecture and start to look into how we can implement caching and CI/CD practices. A plan for integration should be done by WHEN?
* **Make a plan for implementing security and compliance** - Farshid will investigate the security aspects of our environment and application. How do we make it secure and are there any compliance requirements like GDPR that we need to consider? A plan on how to implement security and compliance should be done by WHEN?
* **Develop a prototype of the application** - Ahmed will start developing the web application and should provide us with a prototype latest by WHEN?
* Everyone should write a short report on their findings on their topic and send it to Mattias that will merge it into the final project report.

Our team has set up a WhatsApp group for daily discussions and check-ins. Weekly project meetings have been held, and will be held, on Microsoft Teams. Project progress will be tracked using Trello and all source code will be stored in GitHub.

**REFERENCES**

Fowler, M. 2020. AWS CloudFormation Doesn’t Do That Yet?! No Problem. Medium. Available at: <https://shouldroforion.medium.com/aws-cloudformation-doesnt-do-that-yet-no-problem-fdd1ff01839>. Referenced 26.4.2024.

AWS. 2024. Cognito. Available at: <https://aws.amazon.com/cognito/dev-resources/>. Referenced 26.4.2024.

Wolford, B. 2024. What is GDPR, the EU’s new data protection law?. Available at: <https://gdpr.eu/what-is-gdpr/>. Referenced 26.4.2024.

Biradar, M & Moolya, S. 2020. Integrating with GitHub Actions – CI/CD pipeline to deploy a Web App to Amazon EC2. Available at: <https://aws.amazon.com/blogs/devops/integrating-with-github-actions-ci-cd-pipeline-to-deploy-a-web-app-to-amazon-ec2/>. Referenced 26.4.2024.

AWS. 2024. Well Architected Framework. Available at: https://docs.aws.amazon.com/wellarchitected/latest/framework/general-design-principles.html.

AWS Academy. Cloud Architecting, Module 4 Student Guide. 2022. Available at: <https://awsacademy.instructure.com/courses/76077/modules/items/6836931>. Referenced 26.4.2024.

AWS Academy. Cloud Architecting, Module 5 Student Guide. 2022. Available at: <https://awsacademy.instructure.com/courses/76077/modules/items/6836947>. Referenced 26.4.2024.

AWS Academy. Cloud Architecting, Module 8 Student Guide. 2022. Available at: <https://awsacademy.instructure.com/courses/76077/modules/items/6836980>. Referenced 26.4.2024.

AWS Academy. Cloud Architecting, Module 9 Student Guide. 2022. Available at: <https://awsacademy.instructure.com/courses/76077/modules/items/6836992>. Referenced 26.4.2024.

AWS Academy. Cloud Architecting, Module 10 Student Guide. 2022. Available at: <https://awsacademy.instructure.com/courses/76077/modules/items/6837006>. Referenced 26.4.2024.

AWS Academy. Cloud Architecting, Module 11 Student Guide. 2022. Available at: <https://awsacademy.instructure.com/courses/76077/modules/items/6837019>. Referenced 26.4.2024.

APPENDIX 1

**Instructions for appendices**

Appendices consist of e.g. manuals, questionnaires, tables and figures, and other such materials that do not necessarily need to be included in the actual thesis report. Appendices are placed last in the thesis.

Appendices have no page numbering. Instead they are numbered separately: APPENDIX 1, APPENDIX 2 etc. in the top right-hand corner. If an appendix consists of three pages, for example, they are marked: APPENDIX 2/1, APPENDIX 2/2, APPENDIX 2/3.

When referring to appendices in the text, the page number of the appendix is not mentioned, only the number of the appendix (eg. APPENDIX 1). If referring to a figure in an appendix, mention only the appendix and its number, not the figure. For example: In recent years the number of international students in Finland has grown rapidly (APPENDIX 1).

APPENDIX 2/1

Example of the first page of the multi-paged appendix 2.

APPENDIX 2/2

Example of the second page of the multi-paged appendix 2. When referring to this page, a reference to appendix 2 is used (APPENDIX 2).

APPENDIX 2/3

Example of the third page of the multi-paged appendix 2. Also, when referring to this page, a reference to appendix 2 is used (APPENDIX 2).