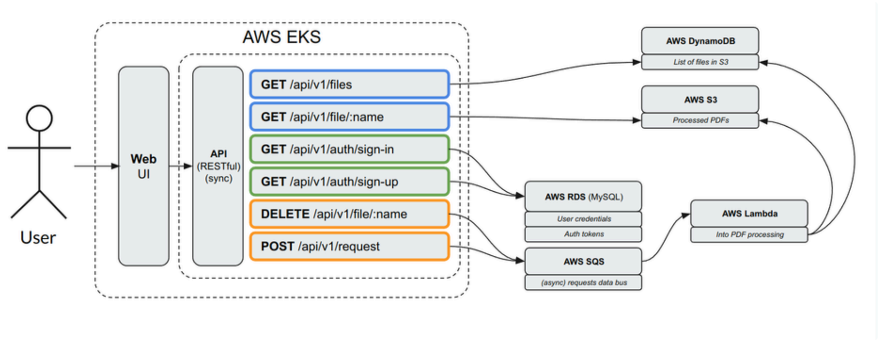
2.0 Application

Contents of the current section:

Application definition and overview

This is a learning application that is designed for those who are learning **DevOps**. The application itself, like the entire project, is intended to cover a broader scope under the hood. The application's main task, consisting of frontend and backend, is to convert requested HTML pages to PDF and save these files with the possibility of having convenient authorized access to these objects through UI. It looks simple, but in the path of the whole operations cycle, we face several services from the world of **AWS: *RDS, SQS Lambda, DynamoDB, and S3*.**

Architecture diagram



Application layers

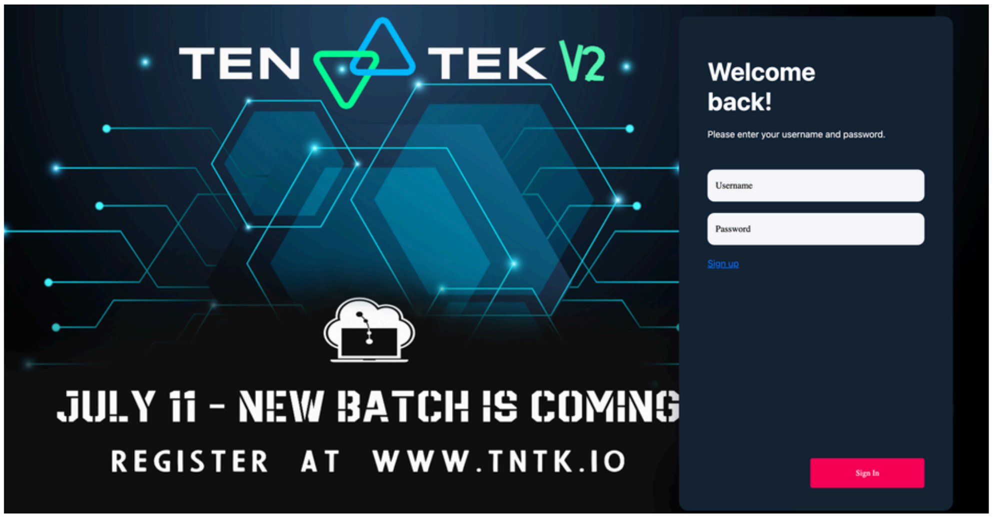
To visualize the entire app scope and understand what is in our application STACK, let's look at it in layers and go through each. We have identified six primary layers, the application of which consists of the frontend, the backend, idempotency, serverless, databases, and object storage.

Frontend

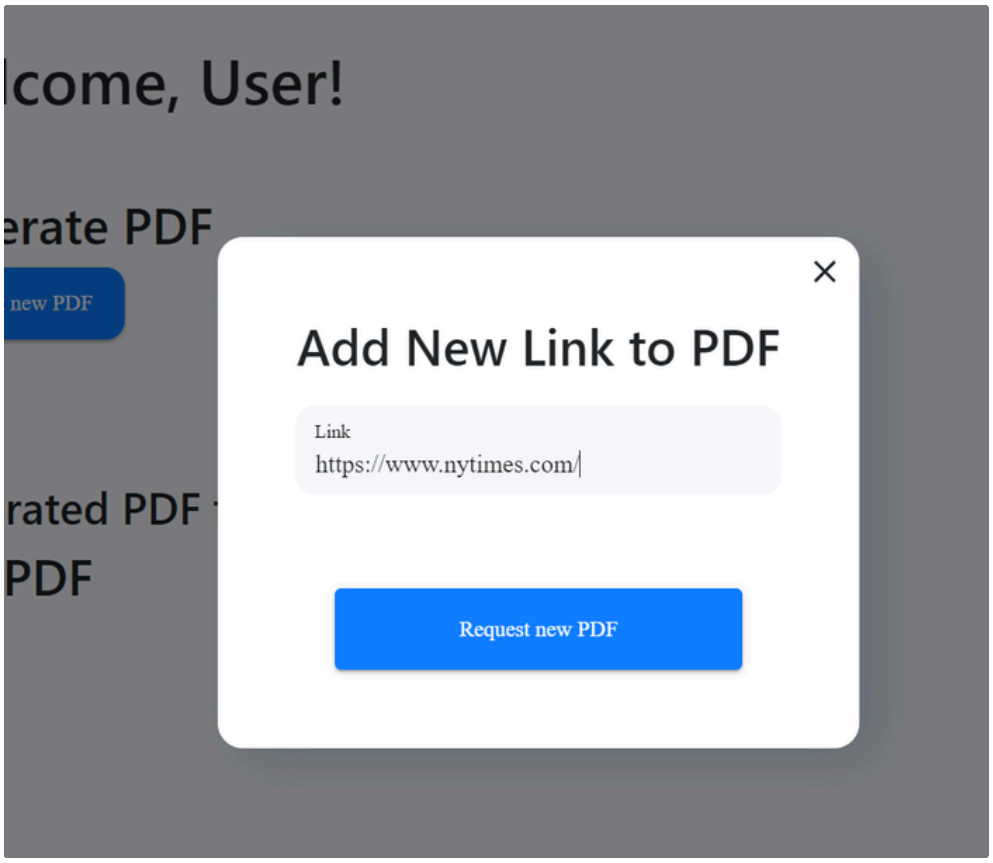
It is a single-page web application written in **react** JavaScript library. The frontend and backend are connected via the **REST API.**

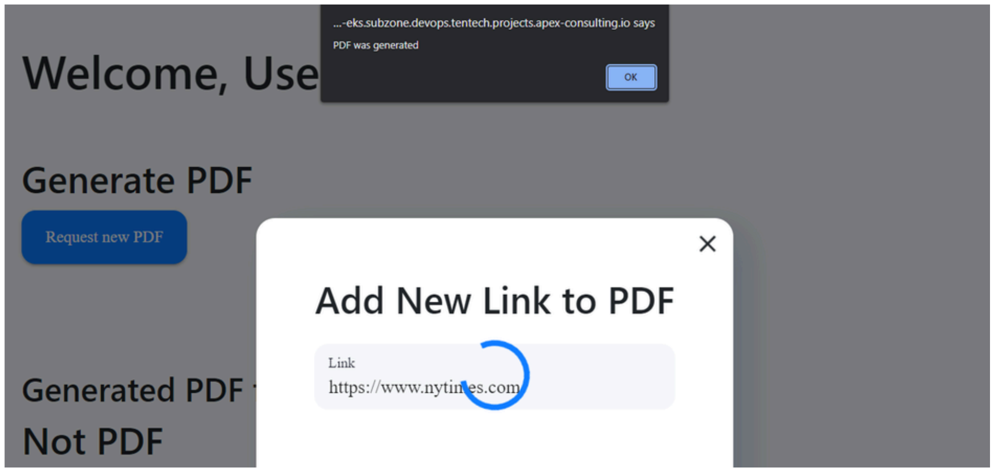
The user interface allows you to perform the following actions with related HTTP methods:

- **Authorization**
  POST: /api/v1/sing-up
  POST: /api/v1/sing-in
- **Execution of PDF conversion request**
  POST: /api/v1/request
- **Removing files from the list.**
  DELETE: /api/v1/file:name
- **List of available files to download**
  GET: /api/v1/files
- **Download specific file**
  GET: /api/v1/file/:name



After authentication, the user will have the option to make a request. After the user presses the corresponding button, a window appears where you can specify the URL to any page on the Internet. After executing the POST (/api/v1/request) request with the URL, a cycle of operations will be performed, eventually returning the result as a pdf file of the requested page.





Let’s look in more detail. When the backend receives a request from the frontend, it will pass the specified URL to the handler, which will render the HTML page and then convert it to a pdf file and save it to the object storage (S3). Then this file will appear in the list of available files for download. The generated file can be downloaded by clicking on the corresponding file name in the list. All files will only be available to users with appropriate access rights.

We will consider the full functionality in the App functionality cycle section. Now let's look at the backend layer.

Backend

Backend provides authentication and initial processing of incoming requests from users through API. It communicates directly with databases (MySQL, Dynamo DB) and storage service S3. It also passes requested URLs from users to Lambda through the SQS service.

Backend part of the application Is written in Golang using 3rt party libraries; the main ones are:

- **Gin Web Framework**. Provides the implementation of RESTful API.
  GitHub - gin-gonic/gin: Gin is a HTTP web framework written in Go (Golang). It features a Martini-like API with much better performance -- up to 40 times faster. If you need smashing performance, get yourself some Gin.
- **GORM. ORM library for Golang.** Provides CRUD operations with databases using known associations and object-relational models.
  GitHub - go-gorm/gorm: The fantastic ORM library for Golang, aims to be developer friendly
- **AWS SDK** with a set of all utilities for programmatically accessing AWS services.
  GitHub - aws/aws-sdk-go-v2: AWS SDK for the Go programming language.
- **AWS Lambda for Go**. Toolset for development of AWS Lambda functions in Golang.
  GitHub - aws/aws-lambda-go: Libraries, samples and tools to help Go developers develop AWS Lambda functions.
- **wkhtmltopdf** command line wrapper. HTML to PDF renderer.
  GitHub - SebastiaanKlippert/go-wkhtmltopdf: Golang commandline wrapper for wkhtmltopdf

There are also other auxiliary libraries in the application. The full list of them you can be found in the file:

https://github.com/tntk-io/tntk-ci/blob/prod/src/api/go.mod

> Please note that the components of the application described above are presented here for reference. It is not necessary to dive deep into their code. We provide this information so that the DevOps engineer understands the stack they are working with and how the application operates. Often, DevOps engineers have to participate in problem-solving with the development team, and their competence and value to the team are enhanced by their programming skills and understanding of architecture. Therefore, we recommend taking a closer look under the hood of the applications you deploy.

Message broker

This is an important component of fault tolerance, especially when it comes to performing heavy tasks such as scraping web pages and converting them to PDFs, as in our case. If we don't use a message broker and a separate handler like Lambda, the backend will be overwhelmed with blocking calls, which could lead to failure and loss of requests.

Amazon SQS (Simple Queue Service) and AWS Lambda are two key services in the Amazon Web Services (AWS) ecosystem often used in building applications with message brokers. We will use them together for build scalable and reliable system.

Amazon SQS provides reliable message delivery, scalability, and management of message processing. It ensures that messages are delivered and processed even if the recipient is temporarily unavailable, and it automatically scales to handle message volume.

The combination of SQS and Lambda offers advantages such as fault tolerance, flexible scalability, and cost reduction. Using SQS as a message buffer and Lambda for message processing creates a resilient architecture capable of handling varying workloads efficiently.

Overall, integrating SQS and Lambda enables the creation of flexible, reliable, and cost-effective applications leveraging message brokers.

Serverless

AWS Lambda is serverless and event-driven compute service that lets you run code for any application or backend service without provisioning or managing servers. Here we can release the workload of the virtual machine and optimize it.

AWS Lambda allows for serverless execution of code, flexible integration with various AWS services including SQS, and pay-per-use pricing, where you only pay for the execution time of your code.

The primary application workload (converting and file storing) is passed to the serverless technology AWS Lambda. It performs several operations:

1. Direct rendering of user-requested HTML page
2. Convert HTML to PDF
3. Save the PDF file into AWS S3
4. Sending file information to NoSQL database AWS DynamoDB.
5. Deleting objects from S3 and records from DynamoDB.

The source code of the Lambda function is stored here:

https://github.com/tntk-io/tntk-ci/blob/prod/src/api/lambda/main.go

Databases

We have been using two types of databases **relational** and **NoSQL**.

MySQL relational database stores the user credentials, which is hosted on the ***AWS RDS*** platform. The backend communicates with MySQL directly without a message queue.

Accordingly, two types of information are stored there:

1. User credentials (Users themselves and their passwords as MD5 hashes)
2. Authorization tokens (simple JWT)

Whenever a user is authorized, a temporary token is created. Accordingly, if the same user is authorized on two different devices, only the last authorized user will work. Previous tokens become invalid.

DynamoDB NoSQL database stores the key value data structure. The key is the user ID, and the value is the list of pdf files ready for download.

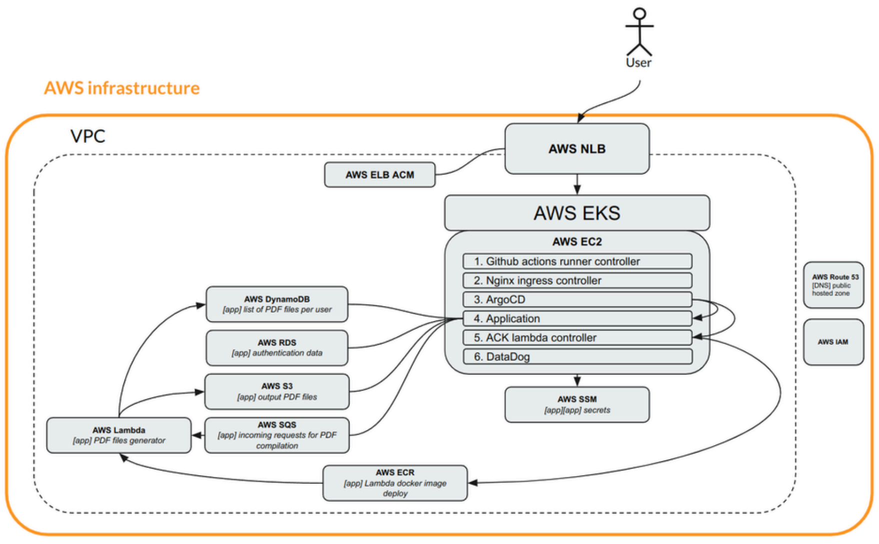
This means that when the API asks for all files available for download files, the backend retrieves the list from DynamoDB and returns it to the user. As a result, the user sees a list of files available for download. When we try to download a specific file, the backend receives a corresponding request on the API then goes to S3, downloads that file, and gives it to the user.

All HTTP GET methods described above communicate with DynamoDB directly opposite to the request POST method when the Lambda function appears in the middle of the file creation process.

Objects storage

Amazon Simple Storage Service (Amazon S3) is an object storage service that is the best and most suitable for storing our files. Every time we generate a PDF it’s stored in the S3 Bucket. Information about the available files in the S3 bucket is obtained from DynamoDB for a specific authorized user.

Application cloud placement diagram

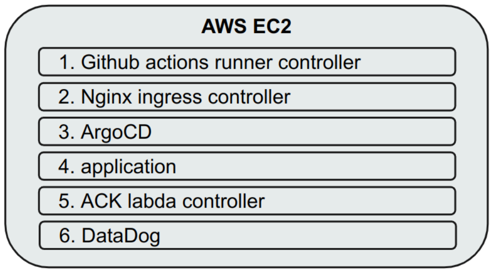


The application lives in the ***EKS*** cluster, which is deployed in ***VPC*** with **a network load balancer**. This type load balancer on AWS, is mandatory requirement for the Ingress-Nginx Controller according to the document.

The network load balancer shifts traffic to our worker node pools. Load balancer terminates SSL connections. Accordingly, it needs a certificate that is stored in ***AWS certificate manager***.

EKS node group

The Node group consists of a minimum of 2 workers. There are 6 main components:



Operational components:

- **Ingress Nginx Controller**: It is a Kubernetes controller that manages the Ingress resources and acts as a reverse proxy to route incoming HTTP and HTTPS traffic to the appropriate services within a cluster. It enables external access to applications running in the cluster.
- **ArgoCD:** It is a declarative continuous delivery tool for Kubernetes. It automates the deployment of applications and provides a GitOps approach, allowing users to define the desired state of applications using Git repositories. ArgoCD ensures that the deployed applications match the desired state defined in the repository.
- **GitHub Actions Runner Controller**: It is a component that manages and coordinates the execution of GitHub Actions workflows. GitHub Actions runners are the compute resources responsible for executing the steps defined in workflows. The runner controller helps manage and scale the runner instances based on the workload.
- **ACK Lambda Controller**: ACK stands for "AWS Controllers for Kubernetes," and the ACK Lambda Controller is a specific implementation within ACK. It is responsible for managing the interaction between Kubernetes clusters and AWS Lambda functions. It allows you to deploy and manage Lambda functions using Kubernetes native resources and tools. This component was specifically chosen to adhere to the **GitOps approach** of deploying applications.
- **Datadog** is a comprehensive monitoring and analytics platform that provides real-time visibility into the performance and health of your infrastructure, applications, and services. It offers a wide range of features including metrics monitoring, log management, application performance monitoring (APM), and more. With Datadog, you can easily collect, analyze, and visualize data from various sources, enabling you to optimize performance, troubleshoot issues, and make data-driven decisions for your organization.

Application component:

- Hosts the **application itself.** There's a backend, a frontend, and a horizontal autoscaler pod.

Parameter Store

We place all the necessary secrets in the **AWS Parameter Store**, which stores all the secrets. Every service, including the application, has access to them. We don’t use k8s secrets manager.