VideoWiki

Installation Guide

Introduction:

The following Installation and Configuration guide provides instructions to Install Videowiki's Software on your servers that are intended to support and run your Videowiki's instance. Including preparing your servers, installing required supporting softwares and integrating with some of the third party vendors that are part of the software.

Pre-Installation Requirements:

The following prerequisites and requirements must be satisfied in order for the installation to be successful:

1. An active server with Linux operating system running (preferably Ubuntu)
2. An active domain name, subdomain support and an active SSL certificate
3. A managed MongoDB cluster, you can always use either MongoDB Atlas (https://www.mongodb.com/cloud/atlas) , Mlab (https://mlab.com/) or deploy your own MongoDB cluster on your servers.
4. An active Amazon Web Services (AWS) account. (https://aws.amazon.com)
5. An active Google Cloud account with billing enabled. (https://cloud.google.com)
6. An active Mailgun account. (https://www.mailgun.com)
7. An active Babbellabs account. (https://babblelabs.com)
8. In case of using the WhatsApp Bot service, having an active Turn.io account (http://turn.io)

Hardware Requirements:

Minimum Hardware Requirements:

- CPU: 4 cores
- RAM: 16 GB
- Hard Drive/Storage: 50GB

Recommended Hardware Requirements:

- CPU: 8 Cores
- RAM: 32GB
- Hard Drive/Storage: 50GB

Software Prerequisites:

Videowiki's components are packaged in docker containers, hence the followings must be installed on the target server:

1- Docker : installation instructions can be found in this documentation https://docs.docker.com/engine/install/

2- Docker Compose: installation instructions can be found in this documentation https://docs.docker.com/compose/install/

Installation and Configuring Procedure:

The installation procedure contains the following steps:

1. Setting up Google Cloud keys and enabling required Google Cloud services
2. Setting up AWS Keys and required AWS S3 buckets.
3. Setting up environment variables for the deployment
4. Installing and configuring the software components

1. Setting up and enabling required Google Cloud Services:

The software uses Google Cloud as its translation service and text-to-speech service, hence requires an active Google Cloud project and enabling some API's in it

1.1: Creating Google Cloud Project:

1. Head to Google Cloud and register a new account (https://cloud.google.com/)
2. Head to Google Cloud console (https://console.cloud.google.com/)
3. Create a new Project by clicking on the "Select a project" dropdown on the top left of the screen and "New Project" in the modal that pops up
4. Enter your project name, take a note for the Project ID and click "Create"
5. after few seconds you should get a notification saying that your project has been successfully created

1.2: Creating credentials to be used with the enabled API's:

(https://cloud.google.com/docs/authentication/getting-started)

1. Go to the Console's home page (https://console.cloud.google.com/)
2. Select your newly created project from the dropdown on the top left
3. Go to "API and services" on the left sidebar
4. Click on the "Credentials" link on the left sidebar
5. you should be able to see "CREATE CREDENTIALS" button on the top center of the screen
6. Click on the "CREATE CREDENTIALS" button and choose "Service account" from the dropdown
7. Choose a name for the service account and click create
8. On the next page, select an owner role for the service account for this project by clicking on the "Select a role" input field, select "Project" and click on "Owner"
9. Click on Continue
10. On the next page, create a private key to be used by clicking on the " + Create key ", then select JSON format and click create. you'll be prompted to download the file key, save it for later use.

1.3: Enabling the Translation, Text-To-Speech and Speech-To-Text API's :

(https://cloud.google.com/endpoints/docs/openapi/enable-api)

1. Go to the Console's home page (https://console.cloud.google.com/)
2. Select your newly created project from the dropdown on the top left
3. Go to "API and services" on the left sidebar
4. A new screen will show up, click on the "Library" link on the left sidebar that will navigate you to the API library
5. In the search box, search for "Cloud translation api"
6. Click on the first search result
7. Enable the API by clicking on the "ENABLE" button
8. Repeat steps 5 through 7 for enabling "Cloud text-to-speech api"
9. Repeat step 5 through 7 for enabling "Cloud speech-to-text api"

2: Setting up AWS S3 buckets and Keys:

2.1: Setting up AWS S3 buckets:

Two buckets should be created, one for storing uploaded videos and assests, and one for transcriptions

1. Start by going to your accounts console (https://console.aws.amazon.com/console/home)
2. open "Services " on the top left and go to "S3"
3. Click on "Create Bucket" button
4. Choose your bucket name and region, click next
5. Enable versioning by click on the checkbox under "Versioning"
6. For the frontend to access the stored items, the bucket should have public access. So uncheck the "Block all public access" checkbox to enable it and click next
7. Click "Create Bucket"
8. Repeat steps 3 through 7 for the second bucket
9. Take note of the created buckets names

2.2: Setting up IAM user

An IAM user is required for the software to be able to access AWS Services

1. Start by going to your accounts console (https://console.aws.amazon.com/console/home)

2. open “Services “ on the top left and go to “IAM”
3. From the left sidebar, click on “Policies”
4. Click on “Create Policy”
5. in the JSON editor, attach the following policy https://gitlab.com/videowiki/videowiki/-/blob/master/aws/user_policy.json

   Note: Don’t forget to change “MEDIA_BUCKET_NAME” and “TRANSCRIPTIONS_BUCKET_NAME” to your corresponding bucket names
6. Click on “Review Policy”
7. Give a name to your policy and note it for later
8. Click on “Create Policy”, the policy should be created successfully after a few seconds
9. From the left sidebar, click on “Users”
10. Click on “Add User” button on the top
11. Choose your user’s name, and under “AWS Access Type”, make sure the “Programmatic access” is checked, then click next
12. in the permissions page, Click on “Attach existing policies directly”
13. Search for the created policy in step 8 and click on the checkbox
14. Click on next then click on “Create User”
15. Download the create .csv credentials file as it contains access key and secret that will be used in environment variables

3. Setting up environment variables for the deployment

All required environment variables can be found in this docker-compose.env file (https://gitlab.com/videowiki/videowiki/-/blob/master/docker-compose.env) , below is a description for each environment variable and it’s purpose

1. AWS_ACCESS_KEY_ID: the access key for the AWS user created in section **2.2.15,** Used to interact with AWS’s transcribe, polly and s3 services
2. AWS_ACCESS_KEY_SECRET: the access key secret corresponding to AWS_ACCESS_KEY_ID
3. AWS_DEFAULT_REGION: the region in which the S3 is deployed

4. AWS_BUCKET_NAME: the name of the bucket used to store the software’s media
5. AWS_TRANSCRIBER_TRANSCRIPTIONS_BUCKET_NAME: the name of the bucket used to store the transcriptions
6. GOOGLE_CLOUD_PROJECT_ID: From the file generated in step **1.2.10,** open the file and find the “project_id” field and copy its value here
7. GOOGLE_CLOUD_CLIENT_EMAIL: From the file generated in step **1.2.10,** open the file and find the “client_email” field and copy its value here
8. GOOGLE_CLOUD_PRIVATE_KEY: From the file generated in step **1.2.10,** open the file and find the “private_key” field and copy its value here
9. FRONTEND_HOST_URL: The domain name on which the software will be deployed behind, including the http/https protocol
10. FRONTEND_HOST_NAME: the hostname only of the deployment
11. FRONTEND_HOST_PROTOCOL: The protocol only of the deployment (http or https)
12. API_ROOT: the location at which the API is deployed (The API external location)
13. WEBSOCKET_SERVER_URL: the location of the websockets service (usually the same as API_ROOT unless the websockets service is deployed somewhere else)
14. VW_SUPER_TRANSCRIBERS_EMAILS: the emails of the users that will act as super transcribers separated by a comma. A super transcriber is a user that would be able to cut videos in most organizations that are assigned to be cut in-house by other organizations
15. BABBLELABS_USERNAME: the username of the babbellabs account, babbellabs is used to support noise canceling on the recorded translations
16. BABBLELABS_PASSWORD: Babbellabs account password
17. MAILGUN_API_KEY: The API key to be used with Mailgun, Mailgun is used as the email client in the software
18. MAILGUN_DOMAIN: the domain registered with Mailgun
19. MAILGUN_ENDPOINT: the endpoint to be used by mailgun, leave empty if the domain is US based
20. DISABLE_PUBLIC_ORGANIZATIONS: whether to disable public user registration or not, in case of yes it’s value should be 1, otherwise leave empty or value of 0. if public registration is set to false, this envrionment variable instructs the app to create a superuser with a default main organization
21. SUPERUSER_EMAIL: the email of the super user in case of disabling public organizations
22. SUPERUSER_PASSWORD: the password of the super user
23. SUPERUSER_ORGANIZATION_NAME: the name of the default main organization

In case the Whatsapp bot will be deployed with the installation, the following environment variables must be filled:

1. VIDEOWIKI_WHATSAPP_NUMBER: the registered Whatsapp number with Turn.io
2. TURNIO_USERNAME: The username of the Turn.io API account
3. TURNIO_PASSWORD: the password of Turn.io API account
4. WHATSAPP_BOT_BREAKVIDEO_WHITELISTED_NUMBERS: The phone numbers of the users that will have access to break videos on WhatsApp separeted by commas

Other environment variables:

1. REDIS_HOST: the host location for where Redis is deployed
2. REDIS_PORT: the port for redis
3. RABBITMQ_SERVER: the connection string for connecting to the rabbitmq cluster

Services Api's Environment Variables:

Services api env variables are the locations at which each service is deployed, in case of deploying on a single server, leaving them to their default values will be fine.

Database Connections environment variables:

Databaset connections environment variables are the connection strings for the database of each service, by default the included docker-compose.yml file spins up a mongodb instance to be used with the installation (NOT PRODUCTION RECOMMENDED), for production usage you can use third party vendors for a managed MongoDB cluster or create your own on your servers.

4. Installing and configuring the software components:

1. Start by ssh-ing to the server on which the installation will be deployed on
2. Make sure you have git, docker and docker-compose softwares installed
3. clone the following repo in a directory called "videowiki" (https://gitlab.com/videowiki/videowiki)
4. move to the project directory (cd videowiki)

5. open the docker-compose.env file and fill your environment variables that will be used
6. modify the nginx configuration file at “nginx_config/default.conf” to reflect your server name
7. Add your SSL certificate files in “nginx_config/” directory with the following namings

   1- Certificate file: nginx-certificate.crt

   2- Key file: nginx-certificate-key.key

8. run “docker-compose up” to spin up the services

5. Deploying on a kubernetes cluster on AWS:

Deploying the software on a kubernetes cluster is dependent on the vendor at which the kube cluster is hosted, either using AWS, GCP or self-managed clusters. The following shows the steps to deploy the software on amazon’s ELK that can be used as a guideline for other vendors as well. Provided with this guide is a directory called “k8s” that includes all the config files required for these steps.

5.1: Creating the kubernetes cluster:

1. Create a VPC for the kube cluster by following this guide using the “Private and public subnets” guide https://docs.aws.amazon.com/eks/latest/userguide/create-public-private-vpc.html
2. Create EKS cluster role by following this guide https://docs.aws.amazon.com/eks/latest/userguide/service_IAM_role.html
3. Create the EKS cluster following this guide https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html

   IMPORTANT NOTE: to be compatible with the remaining of this guide, the kubernetes version to be used is 1.15 with the cluster

4. Install aws-iam-authenticator using the following guide https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html
5. Create a kubeconfig file for Amazon EKS to connect to the kube cluster using the following guide https://docs.aws.amazon.com/eks/latest/userguide/create-kubeconfig.html

   (**note: to use kubectl and create the kubeconfig correctly, you must be logged in with the same user account that created the cluster**)

5.2: Configuring the nodegroups

1. ensure you’re properly connected to the kubernetes cluster and it’s up and running by issuing the command “kubectl get svc”
2. We’ll need to add 4 node groups:

- api nodes: instance type should be t3.medium with volume size of 50GB, minimum of 2 nodes, **kubernetes labels must have service_type: api**

- worker nodes: instance type should be t3.xlarge with volume size 50GB, minimum of 2 nodes, **kubernetes labels must have service_type: worker**

- rabbitmq nodes: instance type should be t3.small with volume size of 20GB, total of 3 nodes, **kubernetes labels must have service_type: rabbitmq**

- redis nodes: instance type should be t3.medium with volume size of 20GB, total of 1 nodes, **kubernetes labels must have service_type: redis**

- elastic stack nodes: instance type should be t3.medium with volume size of 30GB, total of 4 nodes, **kubernetes labels must have service_type: elk**

3. Create EKS node IAM role by following this guide https://docs.aws.amazon.com/eks/latest/userguide/worker_node_IAM_role.html
4. After creating the IAM role, go to the kubernetes cluster dashboard and select the kube cluster.
5. under "Compute" tab, Add a new node group by clicking on "Add Node Group" button to create our first node group
6. name the first nodegourp "api-nodes" or any other preferred name
7. Select the Node IAM Role created from step 3
8. under kubernetes labels, add the following label: service_type: api and click Next
9. under instance type, select t3.medium with disk size 50GB
10. Set your preferred autoscaling values, recommended is minimum 2 maximum 10 desired 2 and click Next
11. under subnets, make sure to select only the private networks (optionally disable remote access to nodes) and click next
12. review the configuration and once ready click "Create"
13. Repeat steps 5 through 12 to create the other node-groups with their respective instance types and kube labels

5.3: Installing Helm

1. Install **helm version 2.16.10** using the following guide https://helm.sh/docs/intro/install/ https://github.com/helm/helm/releases
2. Create tiller service account:

   kubectl -n kube-system create serviceaccount tiller
3. Create role binding:

```
kubectl create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:tiller
```

4. Initialize helm tiller:

   ```
   helm init --service-account tiller
   ```

5.4: Installing Rabbitmq chart

1. Open the provided rabbitmq-1.42.0-ha.values file provided along with this guide to configure the rabbitmq chart
2. find **rabbitmqUsername**, **rabbitmqPassword, managementUsername, managementPassword** keys and change their values accordingly
3. The provided values enables persistent volumes by default and uses the default storage class in the cluster
4. nodeSelector is set to service_type: rabbitmq nodes by default
5. install the helm chart by running:

   ```
   helm install stable/rabbitmq-ha --name rabbitmq --version 1.42.0 --values rabbitmq-1.42.0-ha.values
   ```
6. take a note of the DNS provided for the rabbitmq cluster to be used in environment variables, which will be by default **rabbitmq-rabbitmq-ha.default.svc.cluster.local**

5.5: Deploying Redis

1. Redis is required in cross process communication for websockets, a simple redis pod is sufficient to handle large scale of the use case.
2. Deploy the redis pods by running:

   ```
   kubectl apply -f redis-deployment.yml
   ```

5.6: Deploying VideoWiki services:

1. Open the secrets.txt file and fill the missing values
2. base64-encode the secrets values and generate the secrets file by running “node generate_secrets.js”
3. apply the secrets file by running: kubectl apply -f videowikisecretkeys_secret.yml
4. deploy the services by running the deploy.sh script: bash deploy.sh

5.7: Deploying ingress controller:

1. Please follow this guide to deploy the ingress controller https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html
2. Open the ingress-nginx.yml file and fill the **alb.ingress.kubernetes.io/certificate-arn** field with a certificate arn created through the certificate manger
3. apply the ingress-nginx.yml file by running **kubectl apply -f ingress-nginx.yml**

5.7: Deploying Elastic stack for capturing and storing logs:

1. Provided with this guide a directory k8s/elk that contains configuration files for installing elastic stack for capturing logs and analyzing them
2. You can install it directly by going to the folder "cd k8s/elk" and running the install.sh script "bash install.sh"
3. The pods will be deployed in "elk" namespace

5.7: Deploying Prometheus and Grafana for monitoring application performance:

1. Provided with this guide a directory k8s/monitoring that contains configuration files for installing prometheus and grafana
2. You can install it directly by going to the folder "cd k8s/monitoring" and running the install.sh script "bash install.sh"
3. The pods will be deployed in "monitoring" namespace
4. Take note from the logs of the script for the prometheus server location, which should be **prometheus-server.monitoring.svc.cluster.local**
5. Retrieve your grafana admin password by running **kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo**
6. run the following commands to access the Grafana dashboard
   - export POD_NAME=$(kubectl get pods --namespace monitoring -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=grafana" -o jsonpath="{.items[0].metadata.name}")

   - kubectl --namespace monitoring port-forward $POD_NAME 3000
7. Go to localhost:3000 and Login using the admin password retrieved from step 5

8. go to http://localhost:3000/datasources to add prometheus as datasource
9. click Add Datasource And select Prometheus
10. Add the prometheus server location retrieved from step 4 in **HTTP** section, **URL** field
11. scroll down and click on “Save and Test”, you should get a message saying “Data source is working”
12. To have monitoring dashboards, you can either create one or import already developed dashboards from https://grafana.com/grafana/dashboards
13. Hover over the + sign on the left navbar and click on import
14. under “import via grafana.com” add the dashboard id obtained from the url in step 12
15. We recommend the following dashboards:

    - 11074 for monitoring the VM’s state

    - 10000 for monitoring CPU and ram usage/state per pod