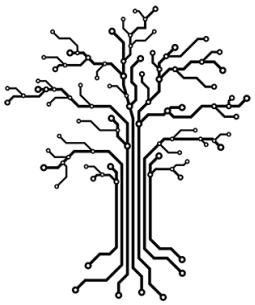
CLIMATETREE



DevOps Documentation

Short Description

The climatetree.wiki setup is hosted on Azure and follows all the underlined rules for a deployment. All the app services and databases have self explanatory names. Our VM has been deployed from a standard template having Linux and it hosts MongoDB.

Intended Audience

Someone who knows how to work with Infrastructure Setup and Resources. They should be well aware of how CI/CD pipeline works and how resources are deployed using a Cloud Service Provider(Azure).

Tools, technologies, and servers used

1. Git: Used for version control.
2. Travis CI: Used for Continuous Integration and deployment.
3. SonarQube: To maintain code quality
4. Docker: To deploy containerized images of microservices
5. MS Azure: Used for hosting all the infrastructure
6. Locust: Used for performance testing

Links to the Hosted Services

Main Site Link (front-end): http://www.climatetree.wiki

Link to all back-end services hosted:

1. Stories Microservice: https://backend-mongo-stories.azurewebsites.net
2. User Microservice: https://usermicroservice-climatetree.azurewebsites.net
3. Places Microservice: https://places-postgres2.azurewebsites.net
4. Geoserver: https://climatetree-geoserver.azurewebsites.net
5. ElasticSearch: https://climatetree-elasticsearch.azurewebsites.net
6. API Gateway: https://climatetree-api-gateway.azurewebsites.net

Databases

Postgre: places-postgresdb.postgres.database.azure.com
MongoDB: hosted on VM on 13.82.198.137 (mongoadmin)

Other components hosted with VM (myLinuxVM):

1. Virtual Network (vnet): myVNET
2. NIC for VM: myNIC
3. Storage Account: fileStorageClimateTree
4. Storage Disk for VM: myLinuxVM_OSDisk
5. Public IP Address: myPublicIP

What happens when you push code to Git?

1. All members of the developer group can either create their own branch and push code onto that or they can push the code to the development branch. No one apart from the DevOps team has the access to push to master branch.
2. The deployment of code into the production environment is done from the master branch, so that access has been restricted to a few users. We didn't want to deploy multiple times.

3. Once a Pull Request (PR) is created for the code to merge into the master, the following things happen:
   i. Reviewers are notified to approve/comment/reject the PR.
   ii. Sonarqube runs code quality gates and only allows the PR to merge if all the checks pass.
   iii. Git checks the repo for any merge conflicts. If there are any merge conflicts, then PR wouldn't be allowed to merge.

4. Once the PR is successfully merged into the master, the image of this code is updated on docker hub.
5. Travis CI then downloads the image and runs it on a virtual environment, if build passes then it is deployed on Azure Cloud containers/app service.

Running Cost:

Microservices	**Appservice plan**	**Cost (per month)**
• Stories • Frontend • Places	B3	147.46$
• Geoserver	P1v2	81.03$
• User • Gateway • Elasticsearch	B3	51.10$

Database	**Hosting**	**Cost (per month)**
Places/User postgres	Postgres azure	272.20$
Stories Mongo	VM	140.16$

Folder Structure

1. SonarQube uses sonar-project.properties. The repository can further be given permissions through GitHub portal and all the configuration can be done Sonarcloud.io dashboard.
2. TravisCI requires travis.yml file
3. Docker requires Dockerfile and docker_push.sh
4. Other conf files may be required based on the technologies used for development. Development guide would shed more light on them.

Known issues and risks

1. Now that MongoDB is hosted on a Virtual Machine, it is not a very scalable solution. Alternate DB available with Azure Cloud is “CosmosDB” and we have already tried using that, we tend to lose some functionality with that and it is not very compatible with ES.
2. We are experiencing slowness issues with Geoserver. It consumes all the processing power of a DB instance. We need a chunkier instance for this to work with no issues.

Future features

1. Using Kubernetes for better management of containers.
2. Adding persistence storage to containers.
3. Moving MongoDB to a scalable service. CosmosDB fixes in future could make it possible.
4. Creation of a dashboard which can run performance test on demand, which may be linked to Locust, or similar tool in background.